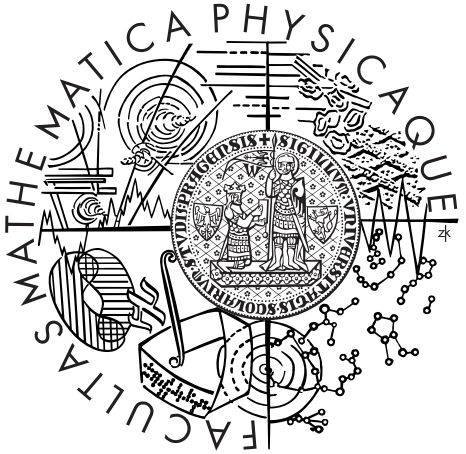
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

DIPLOMOVÁ PRÁCE



Tomáš Benhák

Port HelenOS pro hypervisor Xen

Katedra distribuovaných a spolehlivých systémů

Vedoucí diplomové práce: Mgr. Martin Děcký
Studijní program: Informatika
Studijní obor: Softwarové systémy

Praha 2011

Na tomto místě bych chtěl poděkovat svému vedoucímu Mgr. Martinu Děckému za odborné vedení mé práce, za rady a čas, který mi při vypracování této práce věnoval.

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 6.12.2011 Tomáš Benhák

Název práce: Port HelenOS pro hypervisor Xen

Autor: Tomáš Benhák

Katedra: Katedra distribuovaných a spolehlivých systémů

Vedoucí diplomové práce: Mgr. Martin Děcký

Abstrakt: Cílem práce je port operačního systému HelenOS pro paravirtualizovaný běh pod hypervisorem Xen na platformě IA-32. Výstupem práce je prototypová implementace, která umožňuje běh systému HelenOS jako PV guest pod hypervisorem Xen. Práce analyzuje rozhraní hypervisoru Xen z hlediska paravirtualizovaného operačního systému, který pod hypervisorem běží, relevantní části jádra systému HelenOS a změny v těchto částech, které paravirtualizace vyžaduje.

Klíčová slova: HelenOS, Xen, paravirtualizace

Title: HelenOS port to Xen hypervisor

Author: Tomáš Benhák

Department: Department of Distributed and Dependable Systems

Supervisor: Mgr. Martin Děcký

Abstract: The goal of the master thesis is the paravirtualization of HelenOS operating system for Xen hypervisor on IA-32. The result of the thesis is a prototype implementation which allows to run HelenOS as a PV guest under Xen hypervisor. The thesis analyses the Xen hypervisor interface with respect to the paravirtualized operating system running under it, the relevant parts of HelenOS kernel and changes in them forced by the paravirtualization.

Keywords: HelenOS, Xen, paravirtualization

Obsah

1. Úvod

Virtualizace běhu operačních systémů umožňuje paralelní běh několika instancí operačních systémů v rámci jednoho fyzického stroje. Jedou z používaných praktik je metoda paravirtualizace. Při paravirtualizaci využívá operační systém pro přístup k některým funkcím fyzického stroje rozhraní virtualizačního nástroje. Operační systém tedy musí být pro běh v takovém prostředí upraven – naportován.

HelenOS je experimentální operační systém, který je vyvíjený na Matematicko-fyzikální fakultě Univerzity Karlovy v Praze (zejména v rámci semestrálních a diplomových prací). Tento systém je portován na řadu architektur, jmenovitě AMD64/EM64T (x86-64), ARM, IA-32, IA-64 (Itanium), 32-bit MIPS, 32-bit PowerPC a SPARC V9.

Práce navazuje na [3], v rámci které byla implementována omezená podpora paravirtualizace pro plafotmu Xen v systému HelenOS.

1.1 Cíle práce

Cílem práce je rozšíření operačního systému HelenOS o podporu běhu v paravirtualizovaném prostředí hypervisoru Xen. Práce by se měla skládat ze dvou částí. První částí je analýza rozhraní hypervisoru Xen z pohledu paravirtualizovaného operačního systému. Tato analýza je provedena pro architekturu IA-32. Druhou částí je samotný port operačního systému pro paravirtualizovaný běh pod hypervisorem Xen a popis provedených změn.

1.2 Struktura práce

Text práce je členěn celkem do sedmi kapitol.

Kapitola 2 poskytuje úvod do problematiky virtualizace a popisuje problémy virtualizace na architektuře IA-32.

Kapitola 3 popisuje architekturu a rozhraní hypervisoru Xen z pohledu paravirtualizovaného operačního systému.

Kapitola 4 popisuje části systému HelenOS, které jsou nutné pro pochopení zbytku textu a zdrojových kódů.

Kapitola 5 pohlíží na port z vyšší úrovně, rozebírá metodu implementace paravirtualizace v operačním systému a vlastnosti, které nejsou pro funkční paravirtualizaci nezbytné.

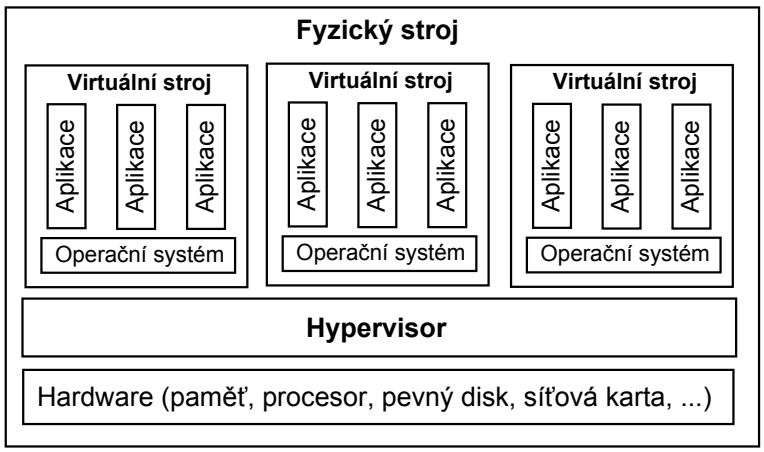
Kapitola 6 popisuje prototypovou implementaci paravirtualizace v operačním systému HelenOS.

Kapitola 7 uzavírá a shrnuje celou práci, včetně možností dalšího rozšíření.

2. Virtualizace

Cílem této kapitoly je stručný úvod do problematiky virtualizace se zaměřením na architekturu IA-32. Virtualizace je však velice obecný pojem. Pro účely této práce se omezíme pouze na techniky virtualizace běhu operačních systémů. Jedná se o techniky, které umožňují běh několika instancí operačních systémů v rámci jednoho fyzického stroje. Rozebrána bude technika úplné virtualizace a paravirtualizace.

Virtualizačnímu nástroji se říká hypervisor nebo také monitor virtuálních strojů. V případě paravirtualizace lze použít rovněž pojem paravirtualizér. Hypervisor má přístup ke skutečnému hardwaru a poskytuje virtuální prostředí pro jednotlivé virtuální stroje. Souvislost hypervisoru a dalších komponent systému zobrazuje obrázek 2.1.



Obrázek 2.1: Souvislost hypervisoru a dalších komponent systému

2.1 Úplná virtualizace

Úplná virtualizace je technika, při které se vytvářejí virtuální stroje, které poskytují ekvivalentní prostředí stroje fyzického. Ekvivalentním prostředím se myslí fakt, že software spuštěný na stroji virtuálním pracuje naprosto stejně, jako kdyby běžel na stroji fyzickém. Z požadavku na ekvivalenci se vynechává otázka časování, kterou ve virtuálním prostředí nelze z pochopitelných důvodů splnit.

Při úplné virtualizaci je velké množství instrukcí prováděno nativně přímo na stroji fyzickém (emulovány jsou typicky pouze instrukce privilegované). Za úplnou virtualizaci tedy není možné považovat například simulaci, při které je každá instrukce simulována softwarově.

2.1.1 Postačující podmínky

Popek a Goldberg uvádějí ve svém článku [1] postačující podmínky úplné virtualizace. V článku jsou formálně popsány charakteristiky fyzického stroje, ze kterých se vychází. Jednou z charakteristik je procesor pracující ve dvou režimech – privilegovaném a uživatelském. Dále jsou instrukce rozděleny do tří množin:

Privilegované instrukce Jsou instrukce, které lze provádět v privilegovaném režimu. Při provedení instrukce v uživatelském režimu je vyvolána výjimka.

Instrukce měnící stav Jsou instrukce, které mění stav zdrojů systému (např. stránkovací tabulky a konfigurační registry).

Instrukce závislé na stavu Jsou instrukce, které se chovají v závislosti na stavu zdrojů.

Jádrem článku je věta: Pokud jsou instrukce měnící stav i instrukce závislé na stavu podmnožinou privilegovaných instrukcí, pak je možná úplná virtualizace.

Z článku také přímo vyplývá metoda implementace monitoru virtuálních strojů metodou "trap and emutate". Pokus o provední privilegované instrukce v uživatelském režimu vyvolá výjimku, která je odchycena monitorem virtuálních strojů a následně odemulována. Jelikož jsou instrukce měnící stav i instrukce závislé na stavu privilegované, mohou být monitorem virtuálních strojů odemulovány. Tak může být zaručena virtualizace všech zdrojů.

2.1.2 Problém úplné virtualizace na IA-32

Intel Pentium obsahuje 17 instukcí, které jsou závislé na stavu, ale nejsou privilegované [2]. Nejsou tedy splněny postačující podmínky úplné virtualizace, viz 2.1.1. Na architektuře IA-32 je přesto možná úplná virtualizace, avšak není možné využít implementace monitoru virtuálních strojů metodou "trap and emulate".

Přepisování instrukcí

Úplnou virtualizaci lze na architektuře IA-32 implementovat pomocí přepisování proudu instrukcí. Tohoto mechanismu využívají např. některé produkty firmy VMware. Virtualizační software musí prozkoumat proud instrukcí a případné použití problémových instrukcí ošetřit.

VM86

Architektura IA-32 zahrnuje možnost virtualizace reálného režimu procesoru, tj. procesoru 8086. Toto je možné prostřednictvím VM86 nebo také virtual real mode. Tento mód byl implementován zejména za účelem paralelního běhu 16-bitových aplikací v 32-bit operačních systémech. Monitorem virtuálních strojů se stáva v tomto kontextu sám operační systém. Obdobný mód pro 32-bit režim však neexistuje.

Intel VT-x a AMD-V

Firmy Intel a AMD nezávisle na sobě vyvinuly rozšíření instrukční sady architektury IA-32, umožňující úplnou virtualizaci za asistence hardwaru. Ačkoli technologie nejsou stejné, pracují na stejném principu. Přidávají další úroveň oprávnění -1 a umožňují vyvolat výjimku u původně problémových instukcí, která je odemulována hypervisorem.

2.2 Paravirtualizace

Paravirtualizace není úplnou virtualizací a neposkytuje stejné prostředí jako stroj reálný. Typicky tedy není možné využít nemodifikovaného kódu a spustit jej v paravirtualizovaném prostředí, avšak prostředí je velice podobné fyzickému. Lze například využít stejný překladač.

Paravirtualizaci lze definovat jako virtualizaci poskytující softwarové rozhraní k některým svým funkcím. Místo problémových instrukcí (těch, kvůli kterým nelze na dané architektuře implementovat úplnou virtualizaci) se tedy využívá rozhraní hypervisoru, který požadovanou operaci provede.

Ne všechen kód je paravirtualizací ovlivněn. Hypervisor poskytuje rozhraní zprostředkovávající operace, které jsou většinou využívány pouze jádrem operačního systému. Aplikace, které pracují v uživatelském režimu, tak mohou ve většině případů zůstat beze změn.

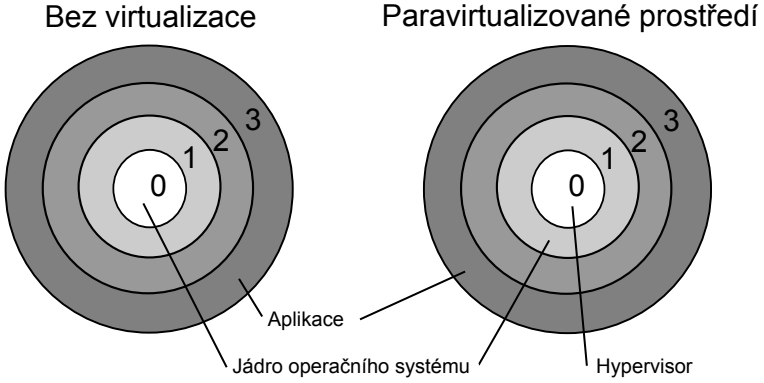
Paravirtualizace má také jinou výhodu. Operační systém si je vědom existence hypervisoru a využívá přímo (často vysoce optimalizované) rozhraní hypervisoru místo emulovaného reálného rozhraní. Tím lze dosáhnout větší rychlosti.

3. Xen hypervisor

Xen hypervisor[1] je virtualizační nástroj podporující architektury IA-32, AMD64, IA-64 a ARM. Xen je primárně paravirtualizované řešení, avšak podporována je rovněž úplná virtualizace pro architektury IA-32 a AMD64 s využitím technologií Intel VT-x a AMD-V[2]. V této kapitole bude popsán Xen hypervisor z pohledu paravirtualizovaného operačního systému na architektuře IA-32, avšak značná část, zejména popis rozhraní, je společná pro všechny Xenem podporované architektury. Popis rozhraní čerpá zejména z veřejných hlavičkových souborů Xenu[3], které obsahují definici všech popisovaných struktur a maker.

3.1 Architektura Xen hypervisoru

Na rozdíl od většiny procesorových architektur, které mají pouze dvě úrovně oprávnění, poskytuje architektura IA-32 úrovně oprávnění čtyři. Moderní operační systémy však využívají pouze úrovně oprávnění dvě – úroveň nula pro jádro operačního systému a úroveň tři pro aplikace. Dvě úrovně tak zůstávají nevyužité. Úroveň jedna a dvě však nejsou místem, kde by bylo možné umístit hypervisor, protože pak by jádro operačního systému běželo na vyšší úrovni oprávnění než samotný hypervisor. Hypervisor je tedy spouštěn na úrovni oprávnění nula a jádro operačního systému je přesunuto na úroveň oprávnění jedna.



Obrázek 3.1: Prostředí nativní vs. paravirtualizované

Posun úrovně oprávnění není však úplně bezproblémový, jelikož na této úrovni oprávnění nezle provádět privilegované operace. Všechny privilegované operace musejí být tedy nahrazeny voláním služeb hypervisoru, kterým se říká hypervolání (z anglického originálu "hypercall"). Princip hypervolání není zcela novou myšlenkou. Stejného principu využívají aplikace při volání služeb jádra operačního systému, těm se říká systémová volání (z anglického originálu "syscall").

Instancím virtuálních strojů se v terminologii Xenu říká domény. První spouštěnou doménou je doména nula, neboli také privilegovaná doména. Ta má výsadní postavení, zejména co se týká přístupu k hardwaru. Další domény jsou domény U, z anglického "unprivileged", tyto nemají přímý přístup k hardwaru.

[1] V době vypracování je aktuální verzí Xen 4.1.1

[2] Podpora Intel VT-x byla přidána ve verzi 3.0.0, podpora pro AMD-V ve verzi 3.0.2

[3] Adresář `xen/include/public/`

Xen je sám o sobě relativně malý kus kódu, který tvoří tenkou vrstvu mezi hardwarem a hostujícím operačním systémem, spravuje základní hardwarové prostředky, avšak sám neobsahuje žádné ovladače ani uživatelské rozhraní. Uživatelské rozhraní, ovladače a nástroje pro správu jsou hlavním úkolem domény nula. Hostujícím operačním systémem pro doménu nula se může stát libovolný operační systém, který byl pro tento účel naportován, nejčastěji Linux.

Jádro domény nula (přesněji jádro hostujícího operačního systému domény nula) je předáno Xenu prostřednictvím zavaděče jako modul. Zavaděč provede spuštění jádra hypervisoru Xen, ten provede veškerou nezbytnou inicializaci a nakonec provede vytvoření a spuštění domény nula (viz bootovací výpis Xenu – příloha C). Doména nula je jediná, která je vytvořena přímo Xenem, vytvoření dalších domén bude provedeno nástroji v doméně nula. Tyto nástroje jsou napsány z velké části v jazyce Python.

3.1.1 Bližší pohled na hypervolání

Pokud vyžaduje hostující operační systém služeb hypervisoru, použije hypervolání. Jedná se o stejný princip, jako když potřebuje aplikace služeb jádra operačního systému, tehdy použije systémová volání. Hypervisor poskytuje přístup k službám, které nelze provést přímo kvůli změně úrovně oprávnění. Zjednodušený scénář hypervolání je následující:

1. Parametry a číslo hypervolání uloží jádro hostujícího operačního systému do registru procesoru a vyvolá softwarové přerušení číslo 82h.
2. Procesor vyvolá obsluhu přerušení (podle IDT). Tabulka IDT je spravována hypervisorem.
3. Obsluha přerušení zjistí číslo hypervolání, které je obslouženo a výsledek je uložen do registru.
4. Po obsluze dojde k návratu z přerušení, operační systém si může přečíst výsledek hypervolání z registru.

Jedinou výjimkou z toho zcénáře je hypervolání iret, které slouží k emulování instrukce iret (návrat z přerušení). První zřejmou změnou je, že se toto volání nikdy nevrátí, stejně jako u klasické instrukce iret. Další změnou je, že stav registru EAX, ve kterém se standardně předává číslo hypervolání, je uložen na zásobníku. Více bude toto hypervolání rozebráno později.

Ačkoli probíhají hypervolání na nejnižší úrovni výše uvedeným způsobem, poskytuje Xen ještě jednu úroveň abstrakce. Jedná se o stránku hypervolání (anglicky "hypercall page"). Stránka hypervolání je naplněna již při vytvoření domény a namapována do virtuálního adresového prostoru.

Každé hypervolání má své unikátní číslo, které je definované ve veřejných hlavičkových souborech Xenu. Číslo hypervolání s názvem `x` lze zjistit makrem `__HYPERVISOR_hypercall_x`. Zjednodušený scénář hypervolání s názvem `x` prostřednictvím stránky hypervolání je následující:

1. Uložení parametrů do registrů.
2. Pomocí instrukce `call` zavolání adresy: `hypercall_page + __HYPERVISOR_hypercall_x * 32`

3. Přečtení návratové hodnoty hypervolání.

Pro každé hypervolání je ve stránce hypervolání vyhrazeno 32B[4], které obsahují následující kód:

```
mov $__HYPERVISOR_hypercall_x, %eax
int $0x82
ret
```

Hypervolání budou v rámci této kapitoly zapisována pomocí prototypů funkcí jazyka C. Názvem funkce bude název hypervolání. Nebude však uváděn návratový typ, ten zpravidla dodržuje konvenci jazyka C (pokud nebude uvedeno jinak) – při úspěchu je vrácena hodnota 0. Prototypy hypervolání jsou převzaty z `extras/mini-os/include/x86/x86_32/hypercall-x86_32.h`.

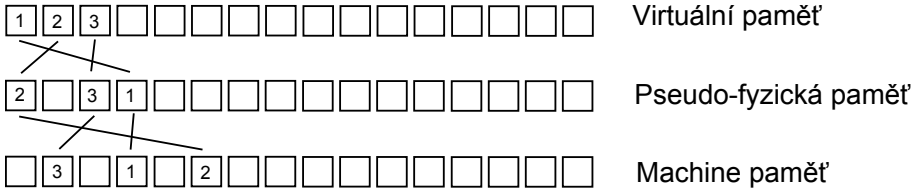
3.1.2 Druhy pamětí

Operační systém fungující v nevirtualizovaném prostředí předpokládá, že má celou fyzickou paměť pouze pro sebe. Toto však není pravda, pokud operační systém běží v paravirtualizovaném prostředí Xenu. Fyzická paměť je sdílena všemi doménami a přidělování fyzických rámců je spravováno hypervisorem Xen. Pod Xenem jsou definovány 3 druhy pamětí (viz obrázek 3.2):

Virtuální Virtuální paměť má v prostředí Xenu stejný význam jako v neparavirtualizovaném prostředí.

Machine Jedná se o skutečné fyzické rámce paměti, v paravirtualizovaném prostředí se jím říká machine rámce, resp. machine adresy.

Pseudo-fyzická Je pomyslnou vrstvou mezi machine a virtuální pamětí. Operační systémy typicky spoléhají na to, že mají souvislou fyzickou paměť začínající na adrese nula, rozdělenou na rámce. K fyzické paměti však nepřistupují přímo, ale mapují rámce do svého virtuálního adresového prostoru. Xen obsahuje převodní tabulky mezi pseudo-fyzickými a machine adresami (P2M i M2P). Pokud mapuje operační systém v nevirtualizovaném prostředí rámec číslo N, bude ve virtuálním prostředí mapovat rámec číslo P2M(N). Jedná se vlastně pouze o očíslování machine rámců domény přístupné přes převodní tabulky, které usnadňuje port operačních systémů. Pseudo-fyzická adresa (nebo číslo rámce) je vyžadována rovněž na mnoha místech rozhraní. Doména je spouštěna s identickým mapováním virtuální paměti, které odpovídá pseudo-fyzické, viz 3.2.1.



Obrázek 3.2: Ilustrace druhů paměti v prostředí Xenu

[4]Při velikosti stránky 4KB je tedy maximální počet hypervolání omezen na 128

3.2 Vytvoření domény

V této sekci bude popsáno výchozí rozvržení paměti domény a také parametry, které poskytuje jádro operačního systému pro hypervisor prostřednictvím ELF poznámek, popř. `__xen_guest` sekce.

3.2.1 Rozvržení paměti

Doména začíná vykonávat svou činnost s daným rozvržením virtuální paměti. Do virtuální paměti jsou souvisle za sebou namapovány následující elementy[5], které tvoří jednu souvislou oblast:

Obraz jádra operačního systému Obraz jádra operačního systému je tvořen sjednocením všech ELF sekcí.

Moduly Formát modulů včetně adresy je uveden ve struktuře start info.

Převodní tabulka P2M Tabulka čísel machine rámců indexovaná číslem pseudo-fyzického rámce.

Start info struktura Struktura je detailně popsána později (viz 3.3.1). Obsahuje základní informace o prostředí.

Stránkovací tabulky Zaváděcí stránkovací tabulky obsahující úvodní mapování.

Zásobník Zaváděcí zásobník (jedna stránka).

Minimálně 512KB nevyužitého prostoru Úvodní alokace se provádí po 4MB. Pokud není za zásobníkem minimálně 512KB volného místa, je úvodní alokace rozšířena o další 4MB.

Tato oblast tvoří rovněž souvislou oblast v pseudo-fyzické paměti a je do virtuálního prostoru namapována na adresu `virt_base` (viz 3.2.2). Umístění oblasti v pseudo-fyzické paměti je určeno nejnižší bázovou adresou všech ELF sekcí mínus `elf_paddr_offset` (viz 3.2.2). Graficky je mapování vyobrazeno na obrázku 3.3.

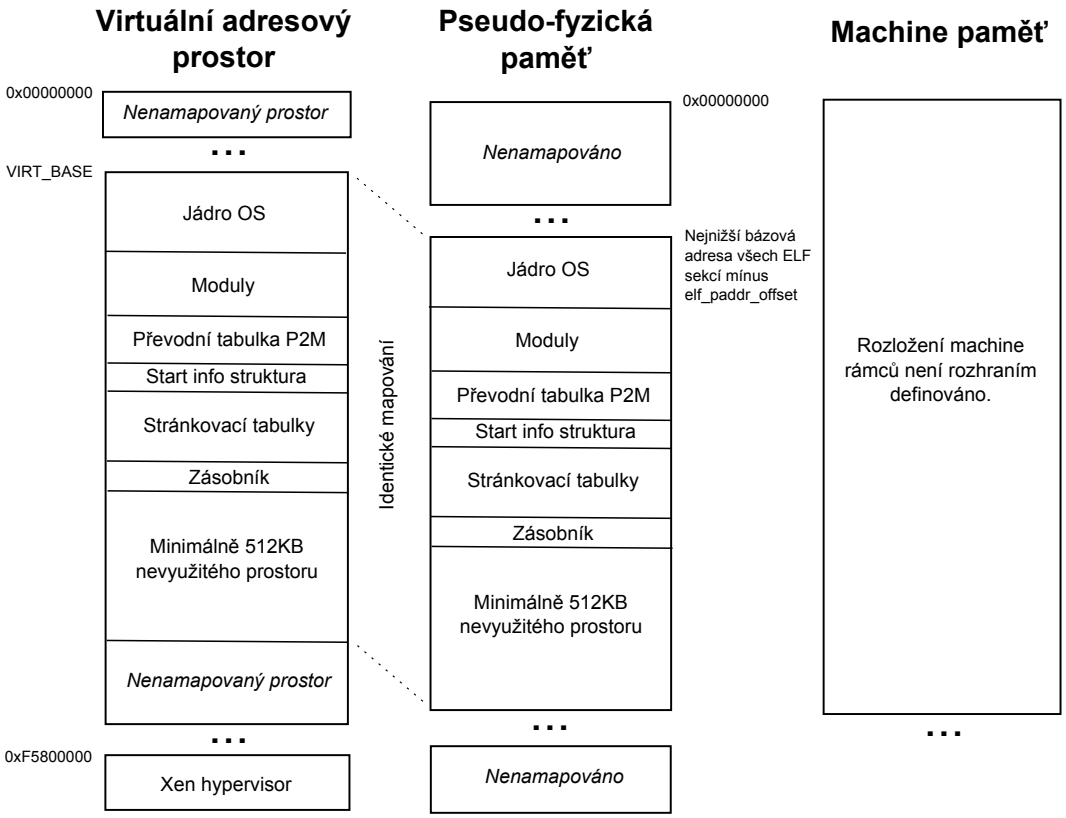
Xen si rovněž v každém virtuálním prostoru vyhrazuje horních 168MB (v případě PAE). Těchto 168MB obsahuje zejména globální převodní tabulku machine rámců na pseudo-fyzické rámce. K ochraně této části paměti před hostujícím operačním systémem je použita segmentace.

3.2.2 Parametry jádra

Aby mohlo být jádro zavedeno prostřednictvím Xenu, musí poskytovat parametry prostřednictvím ELF poznámek (obecně o ELF poznámkách, viz [11]) nebo `__xen_guest` sekce ELF souboru. Obě varianty jsou, co se týče funkce, ekvivalentní a záleží pouze na konkrétním případě, která varianta bude použita.

Xen nejdříve ověřuje, zda jádro neobsahuje ELF poznámky s názvem Xen. Pokud ano, jsou načteny a `__xen_guest` je ignorována. Pokud ELF poznámky s názvem Xen neobsahuje, jsou parametry načteny z `__xen_guest` sekce. Pokud ani ta neexistuje, není možné jádro zavést. Následuje popis možných parametrů:

[5] Převzato z `xen/include/public/xen.h`



Obrázek 3.3: Úvodní mapování vytvořené Xenem

virt_base Adresa, na kterou je namapována úvodní alokace (viz 3.2.1).

elf_paddr_offset Je odečteno od bázových adres ELF sekcí pro získání pseudo-fyzické paměti, na kterou je jádro namapováno.

hypercall_page Číslo stránky ve virtuálním adresovém prostoru, kam bude namapována stránka hypervolání.

xen_ver Verze Xenu, pro kterou je jádro určeno.

loader V současnosti jediná možná volba je generic.

guest_os Název jádra. Slouží pouze k identifikaci jádra.

Pro ELF poznámky jsou ve veřejných hlavičkových souborech Xenu definována makra začínající na `XEN_ELFNOTE`, která přiřazují jednotlivým parametrům číselnou hodnotu. Ta je uvedena jako typ ELF poznámky. Pro více informací viz implementační část.

Xen guest sekce je sekce v ELF souboru s názvem `__xen_guest`, která obsahuje čárkou oddělené dvojce `PARAMETR=hodnota`. Tento řetězec je pak naparsován Xenem při zavádění jádra.

3.3 Info stránky

Info stránky poskytují základní informace o systému. Jejich obsah je definován strukturami `start_info`, resp. `shared_info`[6]. Tyto struktury se vejdou do jedné

[6]Soubor `xen/include/public/xen.h`

stránky a jsou rovněž zarovnány na její začátek, proto lze k jednotlivým položkám přistupovat přetypováním adresy začátku dané stránky na ukazatel struktury definující obsah stránky.

3.3.1 Start info

Start info stránka poskytuje základní informace o systému, které se za běhu nemění. Je namapována Xenem do adresového prostoru domény. Její adresa je předána doméně prostřednictvím registru ESI. Význam jednotlivých položek shrnuje následující přehled:

magic Položka `magic` obsahuje řetězec obsahující verzi hypervisoru a platformu ve formátu xen-verze-platforma.

nr_pages Položka `nr_pages` určuje celkový počet rámců, které jsou pro doménu vyhrazeny, neboli přidělená velikost operační paměti, kde jednotkou je jeden rámec.

shared_info Položka `shared_info` je machine adresa shared info stránky, která bude probrána později.

flags Přes položku flags je doméně předáno několik údajů. Jednotlivé flagy jsou definovány pomocí maker začínajících na `SIF` (zkratka za Start Info Flag). Jestli je některý flag aktivní, lze zjistit standardně pomocí bitového operátoru OR. První flag `SIF_PRIVILEGED` určuje, zda se jedná o privilegovanou doménu, tedy doménu nula. Další flag `SIF_MULTIBOOT_MOD` se týká předávaných modulů a určuje, jestli splňuje specifikaci multiboot. Flag `SIF_MOD_START_PFN` určuje, jestli je položka `mod_start` virtuální adresa nebo číslo pseudo-fyzického rámce.

store_mfn, store_evtchn Sdílená stránka a kanál událostí pro komunikaci s databází Xenstore. Bude probráno později (viz 3.8.2).

console Obsah se liší dle toho, zda-li je doména privilegovaná. Pro neprivilegovanou doménu obsahuje sdílenou stránku a kanál událostí umožňující přístup ke konzoli.

pt_base Virtuální adresa zaváděcích stránkovacích tabulek. Ukazuje na tabulku nejvyšší úrovně.

nr_pt_frames Celkový počet rámců, které zabírají zaváděcí stránkovací tabulky.

mfn_list Ukazatel na převodní tabulku P2M.

mod_start Virtuální adresa začátku předaného modulu. Pokud je nastaven flag `SIF_MOD_START_PFN`, potom se jedná o číslo pseudo-fyzického rámce.

cmd_line Příkazová řádka jádra z konfiguračního souboru. Pomocí této příkazové řádky lze předat jádru různé parametry.

3.3.2 Shared info

Shared info stránka poskytuje informace, které se za běhu systému dynamicky mění. Na rozdíl od start info stránky není shared info stránka namapována do virtuálního adresového prostoru již při spuštění domény. O namapování shared

info stránky se musí postarat sama doména. Machine adresa shared info stránky je uvedena ve stránce start info.

Shared info stránka poskytuje informace o virtuálních procesorech, kanálech událostí a informaci o aktuálním čase. Jednotlivé položky budou popsány detailně až v části věnující se dané problematice. Pro přehled je uveden celý obsah shared info struktury, včetně všech vnořených struktur:

vcpu_info[] Pole obsahující informace o jednotlivých virtuálních procesorech. Indexováno číslem virtuáního procesoru, obsah definován strukturou `vcpu_info`.

evtchn_upcall_pending viz 3.6.2.

evtchn_upcall_mask viz 3.6.2.

evtchn_pending_sel viz 3.6.2.

arch Část specifická pro architekturu IA-32.

cr2 Virtualizovaný registr CR2.

time Údaje týkající se práce s časem, blíže popsáno v sekci 3.7. Obsah definován strukturou `vcpu_time_info`.

version viz 3.7

tsc_timestamp viz 3.7

system_time viz 3.7

tsc_to_system_mul viz 3.7

tsc_shift viz 3.7

evtchn_pending viz 3.6.2.

evtchn_mask viz 3.6.2.

wc_version viz 3.7.

wc_sec viz 3.7.

wc_nsec viz 3.7.

arch Část specifická pro architekturu IA-32.

max_pfn Maximální číslo pseudo-fyzického rámce.

pfn_to_mfn_frame_list_list Obsahuje číslo rámce, který obsahuje čísla rámců použitých pro převodní tabulku P2M.

3.4 Správa paměti

Stránkovací tabulky slouží k převodu virtuálních adres na adresy fyzické. Xen poskytuje doménám iluzi fyzické paměti, které se říká paměť pseudo-fyzická. Avšak tato iluze je implementována pouze softwarově pomocí převodních tabulek a procesor jí nerozumí. Do stránkovacích tabulek, jelikož ty jsou přímo využívány procesorem, se tedy musí jako adresa fyzických rámců zapsat přímo machine adresa.

Pokud by mohla doména zapisovat libovolně do svých stránkovacích tabulek, mohla by si do svého adresováho prostoru namapovat libovolný kus paměti, přestože patří jiné doméně nebo samotnému hypervisoru. Z tohoto důvodu lze stránkovací tabulky namapovat do adresového prostoru domény pouze pro čtení.

Veškeré zápisy do stránkovacích tabulek a tabulek deskriptorů segmentů jsou kontrolovány Xenem, v případě porušení bezpečnosti je operace zamítnuta. V této sekci budou rozebrány metody, kterými Xen kontroluje zápisy do stránkovacích tabulek a tabulek deskriptorů segmentů.

3.4.1 Tabulky přímo zapisovatelné

Přímo zapisovatelné stránkovací tabulky vytváří částečně iluzi, že lze do stránkovacích tabulek přímo zapisovat, avšak zapsané adresy jsou opět machine adresy. Operační systém používající přímo zapisovatelné stránkovací tabulky tedy musí nejprve provést převod na machine adresu.

Tato metoda byla ve starších verzích Xenu výhodná zejména pro hromadné aktualizace stránkovacích tabulek, což bylo dáno metodou implementace. Při pokusu o zápis do stránkovací tabulky byla výjimka zachycena Xenem, příslušná část stránkovací tabulky byla označena jako zapisovatelná a zároveň odpojena z hierarchie. Doména poté mohla provést všechny úpravy, Xen poté ověřil korektnost úprav a upravenou část opět připojil do stránkovacích tabulek (viz [6]).

Nové verze Xenu však již neprovádí odpojování stránkovacích tabulek[7]. Pro dosažení iluze zapisovatelných stránkovacích tabulek se využívá emulace instrukcí. Ve chvíli, kdy se pokusí doména zapsat do stránkovací tabulky, je vyvolána výjimka (stránkovací tabulky jsou namapované pouze pro čtení). Tato výjimka je odchycena Xenem a v rámci zpracování je instrukce, která provádí zápis, odemulována, což Xenu dovoluje ověřit korektnost a případně zamítnutí operace.

Přímo zapisovatelné stránkovací tabulky lze povolit hypervoláním `vm_assist`, které slouží k povolení nebo zakázání různých vlastností dle zadaných parametrů. Pro povolení zapisovatelných stránkovacách tabulek se jako první parametr použije `VMASST_CMD_enable` a jako druhý parametr `VMASST_TYPE_writable_pagetables`. Obě tato makra jsou definována ve veřejných hlavičkových souborech Xenu.

3.4.2 Plně paravirtualizovaná varianta

Plně paravirtualizovaná verze stránkovacích tabulek vyžaduje každou změnu provést prostřednictvím hypervolání. Xen ověří korektnost operace, a pokud není operace korektní, je zamítnuta.

Úpravy stránkovacích tabulek se provádějí pomocí hypervolání:

```
mmu_update(mmu_update_t *req, int count, int *success_count,
           domid_t domid)
```

Hypervolání umožňuje provést aktualizaci více položek stránkovacích tabulek najednou. Parametr `req` je pole požadavků (viz ukázka 3.1), pomocí nichž se specifikuje, jaké položky stránkovacích tabulek se mají aktualizovat.

Ukázka 3.1: Požadavek pro aktualizaci záznamu ve stránkovací tabulce (soubor `xen/include/public/xen.h`)

```
struct mmu_update {
    uint64_t ptr;       /* Machine address of PTE. */
```

[7] Vyvozeno na základě pozorování zdrojových kódu hypervisoru Xen 4.1.1

```
    uint64_t val;       /* New contents of PTE.     */
};
```

Položka `ptr` (struktury `mmu_update`) určuje machine adresu záznamu ve stránkovacích tabulkách, který se má aktualizovat. Položka `val` (struktury `mmu_update`) určuje novou hodnotu tohoto záznamu. Počet požadavků je předán parametrem `count`. Parametr `success_count` vrací počet úspěšně provedených požadavků. Xen přímo neříká, které požadavky se nepovedly, zjištění je odpovědností domény. Poslední parametr hypervolání říká, ve které doméně se má změna provést. Neprivilegovaná doména však může provádět změny pouze ve svém adresovém prostoru a uvede tedy `DOMID_SELF`.

Pro aktualizaci stránkovacích tabulek slouží rovněž hypervolání `update_va_mapping(unsigned long va, uint64_t val, unsigned long flags)`, které aktualizuje položku aktuálně instalovaných stránkovacích tabulek, která mapuje virtuální adresu zadanou jako první parametr `va`. Nová hodnota se předává pomocí druhého parametru `val`[8].

3.4.3 Připíchnutí typu rámce

Xen eviduje u každého machine rámce jeho typ, který určuje, jakým způsobem je rámec využit. Typ rámce je vzájemně se vylučující, tzn., že jeden rámec je vždy právě jednoho typu. Možné typy rámců jsou[9]:

- Stránkovací tabulka první až čtvrté úrovně (pro každou úroveň jeden typ)
- Rámec použit jako GDT/LDT
- Sdílená stránka
- Rámec se zapisovatelným mapováním
- Stránka bez speciálního použití

U každého rámce Xen eviduje také počet referencí, tj. kolikrát je jako daný typ použit. Pokud se počet referencí zvýší z nuly na jedna, provádí Xen validaci obsahu. Validace se provádí například při každé instalaci nových stránkovacích tabulek, což se děje především při přepínání kontextu.

Aby se předešlo nadbytečné validaci, umožňuje Xen rámec připíchnout (z anglického "pin"). Připíchnutí v podstatě znamená zvýšení počtu referencí o jedna, tzn., že počet referencí nikdy neklesne na nulu a Xen, v případě znovupoužití rámce, nebude muset provádět novou validaci obsahu.

Připíchnout rámec lze pouze jednou – nefunguje rekurzivně. Připíchnutí již připíchnutého rámce nemá žádný efekt.

3.4.4 Různé operace nad MMU

Hypervolání `mmuext_op(struct mmuext_op *op, int count, int * sccess_count, domid_t domid)` je určeno k provádění různých operací nad MMU. Prvním parametrem je pole operací[10]. Druhý parametr určuje počet operací. Třetím

[8] Na IA-32 je parametr rozdělen na dolích a horních (v tomto pořadí) 32 bitů a je předáván pomocí dvou parametrů

[9] Jednotlivé typy jsou definovány pomocí maker v souboru `xen/include/asm-x86/mm.h`

[10] Struktura `mmuext_op` je definována v `xen/include/public/xen.h`

parametrem je vrácen počet operací, které byly úspěšně provedeny. Poslední parametr určuje id domény, neprivilegovaná doména uvede `DOMID_SELF`. Nebudeme zde uvádět všechny podporované operace, avšak pouze ty, které jsou pro práci relevantní[11].

MMUEXT_PIN_{L1-L4}_TABLE Slouží k přípíchnutí typu rámce. Číslo machine rámce se předává položkou `arg1.mfn` struktury `mmuext_op`.

MMUEXT_UNPIN_TABLE Slouží k "odpíchnutí" typu rámce. Číslo machine rámce se předává položkou `arg1.mfn` struktury `mmuext_op`.

MMUEXT_NEW_BASEPTR Slouží k instalaci nových stránkovacích tabulek. Číslo machine rámce stránkovací tabulky nejvyšší úrovně se předává položkou `arg1.mfn` struktury `mmuext_op`.

Prostřednictvím hypervolání `mmuext_op` lze rovněž spravovat obsah TLB. Slouží k tomu následující operace:

MMUEXT_TLB_FLUSH_ALL Provede zneplatnění celé TLB na všech virtuálních procesorech.

MMUEXT_INVLPG_ALL Provede zneplatnění jedné adresy na všech virtuálních procesorech.

MMUEXT_INVLPG_MULTI Provede zneplatnění jedné adresy na zvolených virtuálních procesorech. Procesory lze vybrat prostřednictvím bitové mapy `vcpumask`.

MMUEXT_TLB_MULTI Provede zneplatnění celé TLB na zvolených virtuálních procesorech. Procesory lze vybrat prostřednictvím bitové mapy `vcpumask`.

3.4.5 Segmenty

Stejně jako změny ve stránkovacích tabulkách jsou i změny v GDT a LDT spravovány Xenem. Xen poskytuje výchozí tabulku GDT, která obsahuje flat segmenty, takže operační systémy, které segmenty přímo nevyužívají, nebudou muset instalovat své tabulky deskriptorů. Selektory výchozích segmentů do GDT jsou definovány makry[12]:

FLAT_KERNEL_CS Kódový segment jádra.

FLAT_KERNEL_DS Datový segment jádra.

FLAT_KERNEL_SS Stejný jako DS.

FLAT_USER_CS Kódový uživatelský segment.

FLAT_USER_DS Kódový datový segment.

FLAT_USER_SS Stejný jako DS.

Xen si vyhrazuje horních 168MB paměti v každém adresovém prostoru (při zapnutém PAE, nové verze Xenu však podporu PAE vyžadují). Pro ochranu této paměti je využita segmentace, takže žádný segment využívaný doménou se nesmí

[11] Všechny operace lze nalézt v souboru `xen/include/public/xen.h`

[12] Soubor `xen/include/public/arch-x86/xen-x86_32.h`

krýt s touto oblastí. Všechny výše uvedené segmenty začínají na adrese nula a mají velikost 3928MB (4GB mínus 168MB).

Nová GDT tabulka se nastavuje hypervoláním `set_gdt(unsigned long *frame_list, int entries)`. Parametr `frame_list` je pole čísel machine framů, které mají být použity jako GDT. Parametr `gdt_items` určuje počet záznamů GDT tabulky (bez výchozích segmentů).

Aktualizace jedné položky GDT tabulky se provede hypervoláním `update_descriptor(uint64_t addr, uint64_t val)`. Parametr `addr` říká machine adresu deskriptoru, který se má aktualizovat. Parametr `val` určuje novou hodnotu deskriptoru[13].

3.4.6 Sdílená paměť

Sdílená paměť je zpřístupněna pomocí tzv. grant tabulek. Každá doména má svou grant tabulku, ve které může specifikovat oprávnění jiných domén ke svým rámcům. Pokud pomocí grant tabulky poskytne jiné doméně oprávnění ke svému rámci, ta si jej pak může namapovat do svého adresového prostoru. Navíc lze specifikovat i oprávnění, se kterým si může doména rámec namapovat.

Každý záznam v grant tabulce je identifikován tzv. grant referencí. Tuto referenci lze například zveřejnit v databázi Xenstore, odkud si ji protistrana přečte a provede namapování. Pro práci s grant tabulkami je určeno hypervolání `grant_table_op(unsigned int cmd, void *uop, unsigned int count)`. Prvním parametrem hypervolání je prováděná operace (v jednom hypervolání je možné provést více stejných operací s různými parametry). Druhým parametrem hypervolání se předává pole struktur, které obsahují parametry prováděných operací. Poslední parametr hypervolání určuje počet operací.

Základní práce s grant tabulkami

Nastavení grant tabulek domény je provedeno prostřednictvím operace `GNTTABOP_setup_table`. Parametry operace jsou definovány strukturou typu `gnttab_setup_table_t`. Pomocí struktury se definuje, pro kterou doménu jsou grant tabulky vytvářeny, počet rámců a ukazatel na místo, kam hypervisor vyplní čísla machine rámců, které jsou pro grant tabulky použity.

Tyto tabulky lze namapovat do adresového prostoru zapisovatelné a aktualizace probíhá bez asistence hypervisoru. Samotné ověření korektnosti je provedeno až při namapování vzdálenou doménou. Záznam grant tabulky je definován strukturou `grant_entry`[14]. Každý záznam obsahuje číslo sdíleného machine rámce, id domény, pro kterou je sdílení povoleno, a flagy, pomocí kterých se určí povolené operace.

Pro namapování sdíleného rámce je určena operace `GNTTABOP_map_grant_ref`. Ukázka 3.2 zobrazuje strukturu definující parametry této operace. Vstupními parametry jsou `host_addr`, což je pseudo-fyzická adresa v prostoru volající domény, kam bude paměť namapována. Parametry `ref` a `dom` určují doménu a grant referenci sdílené paměti. Položka `handle` bude po návratu obsahovat hodnotu, která

[13] Hypervolání má ve skutečnosti parametry 4, jelikož 64-bitové hodnoty se rozdělují na dolních a horních 32 bitů (v tomto pořadí).

[14] Soubor `xen/include/public/grant_table.h`

je použita v případě odmapování operací `GNTTABOP_unmap_grant_ref`. Položkou `dev_bus_addr` je vrácena adresa, která může být použita vstupně/výstupním zařízením k adresaci sdíleného rámce. Možné hodnoty položek `flags` a `status` jsou uvedeny v souboru `xen/include/asm-x86/grant_table.h`, kde jsou rovněž uvedeny další operace a popis jejich parametrů.

Ukázka 3.2: Parametry operace `GNTTABOP_map_grant_ref` (soubor `xen/include/asm-x86/grant_table.h`)

```
struct gnttab_map_grant_ref {
    /* IN parameters. */
    uint64_t host_addr;
    uint32_t flags;               /* GNTMAP_* */
    grant_ref_t ref;
    domid_t  dom;
    /* OUT parameters. */
    int16_t  status;              /* GNTST_* */
    grant_handle_t handle;
    uint64_t dev_bus_addr;
};
```

3.5 Multiprocesor

Xen obsahuje podporu pro více procesorů od svého prvopočátku. Pro neprivilegovanou doménu lze počet procesorů nastavit v konfiguračním souboru. Při spuštění domény je však aktivní pouze jeden procesor, zbylé je třeba inicializovat a spustit pomocí hypervolání. Virtuální procesor je třeba nejprve inicializovat, tj. nastavit kontext, poté je možné jej spustit.

Pro práci s virtuálními procesory je určeno hypervolání `vcpu_op(int op, int vcpuid, void *extra_args)`. První parametr je operace, která se má provést na virtuálním procesoru, jehož id je předáno v druhém parametru. Pokud vyžaduje daná operace nějaká data, jsou předána v posledním parametru.

3.5.1 Inicializace procesoru

Virtuální procesor je inicializován operací `VCPUOP_initialise`. Kontext procesoru je předán jako poslední parametr hypervolání a je reprezentován strukturou `vcpu_guest_context`[15]. Struktura reprezentuje kompletní kontext virtuálního procesoru, jedná se zejména o stav registrů, obsah trap tabulky a callbacky pro zpracování událostí.

3.5.2 Spuštění

Jakmile je procesor inicializován, je možné jej spustit. Ke spuštění procesoru slouží operace `VCPUOP_up`. Protože operace nemá žádná data, je jako poslední parametr uveden nulový ukazatel.

[15] Soubor `xen/include/public/arch-x86/xen.h`

3.6 Systém přerušení

V prostředí Xenu existují dva mechanismy přerušení – trapy (z anglického "traps") a události. Trapy jsou analogické systému přerušení z architektury IA-32 a jsou určeny pro zpracování výjimek. Události jsou čistě softwarová implementace přerušení Xenu.

3.6.1 Trapy

Trapy jsou zpracovávány prostřednictvím tzv. trap tabulky (obdoba IDT z architektury IA-32). Jednotlivá přerušení jsou číslována pomocí vektorů, přičemž ty mají analogický význam jako vektory na architektuře IA-32. Rovněž formát trap tabulky je velice podobný tabulce IDT. Ukázka 3.3 zobrazuje strukturu definující položku trap tabulky.

Ukázka 3.3: Struktura definující položku trap tabulky (soubor `xen/include/public/arch-x86/xen.h`)

```
struct trap_info {
    uint8_t       vector;
    uint8_t       flags;
    uint16_t      cs;
    unsigned long address;
};
```

Položka `vector` určuje číslo přerušení. Číslo přerušení tedy není určeno indexem v tabulce a index také nehraje žádnou roli. Přes položku `flags` se předávají dva údaje. První až třetí bit určuje minimální úroveň oprávnění, ze které lze přerušení vyvolat. Čtvrtý bit určuje, zda jsou při zpracování přerušení maskovány události. Nastavený bit znamená, že jsou události maskovány a jejich zpracování je tedy odloženo.

Trap tabulka je instalována hypervoláním `set_trap_table(trap_info_t *table)`, jediným parametrem je virtuální adresa trap tabulky.

Systémová volání

Speciální význam v trap tabulce má vektor `0x80`. Tento vektor se zpravidla v operačních systémech využívá pro systémová volání, a proto také Xen předpokládá, že bude k tomuto účelu využit. Zdrojový kód Xenu obsahuje pro tento vektor mnoho výjimek, aby bylo zpracování co možná nejefektivnější.

3.6.2 Události

Události jsou generickým systémem přerušení Xenu. Lze je rozdělit do tří kategorií:

Mezidoménová komunikace Jsou události sloužící ke komunikaci mezi doménami.

Fyzické IRQ Jedná se o události, které jsou napojeny na fyzické IRQ. Jsou využívány například doménou nula pro přístup k přerušení od fyzických zařízení.

Virtuální IRQ Jsou události, které reprezentují IRQ virtuálních zařízení (např. časovač).

Zpracování událostí

Aby mohla doména dostávat události, musí v první řadě zaregistrovat callbacky, které slouží k jejich zpracování. To lze provést pomocí hypervolání:

```
set_callbacks(
 unsigned long event_selector, unsigned long event_address,
 unsigned long failsafe_selector, unsigned long failsafe_address)
```

Pomocí hypervolání jsou nastaveny dva typy callbacků, pro každý je nastaven segment a adresa, která bude Xenem zavolána. První registrovaný callback je určen pro "normální" zpracování událostí. Druhý "failsafe" callback je volán v případě, že není možné vyvolat normální zpracování v důsledku nějaké chyby.

Dále je třeba určit, jaké události budou doručovány. Pro doručování událostí z určitého zdroje je nezbytné:

1. Vytvořit lokální port pro doručování událostí.
2. Spojit port prostřednictvím kanálu se zdrojem událostí. Zdrojem událostí může být fyzické nebo virtuální IRQ, případně port otevřený v jiné doméně (mezidoménová komunikace).

Zjednodušený scénář zpracování události je následující:

1. Zdroj generuje událost, ta je Xenem zapsána do událostí čekajících na zpracování v shared info stránce.
2. Je proveden tzv. upcall (pokud není doručení maskováno), tedy vyvolán callback pro zpracování události na procesoru, se kterým je daný kanál spojen.
3. Callback přečte z shared info stránky všechny události čekající na zpracování pro daný virtuální procesor a zpracuje je.
4. Prostřednictvím hypervolání `iret` je proveden návrat z přerušení.

Zjištění, které události čekají na zpracování, není z důvodu optimalizace úplně přímé. Xen poskytuje bitovou mapu (`evtchn_pending` ve struktuře `shared_info`) společnou pro všechny procesory. Pokud je na port číslo N doručena událost, je nastaven N-tý bit na hodnotu jedna. Aby bylo urychleno zpracování, je rovněž nastaven odpovídající selektor (odděleně pro každý virtuální procesor), který tvoří index do bitové mapy.

Doručení událostí lze "maskovat", což v podstatě znamená pouze to, že pro maskovaný kanál nebude proveden upcall. Mapa maskovaných kanálů je součástí struktury `shared_info`, jedná se o položku `evtchn_mask`. Tento mechanismus dovoluje například zpracovávat některé kanály pomocí pollingu, což může být někdy výhodnější.

S každým virtuálním procesorem je spojena další položka, jedná se o `evtchn_upcall_pending`. Tato položka je nastavena Xenem na hodnotu 1 vždy, když je nebo má být (ale je maskován pomocí `evtchn_upcall_mask`) vyvolán upcall. Hostující operační systém musí tuto položku vynulovat v rámci callbacku před tím, než zkontroluje, které události čekají na zpracování, aby nedošlo k "set-and-check" race condition.

Hypervolání

Pro práci s kanály událostí je určeno hypervolání `event_channel_op(int cmd, void *op)`. Prvním parametrem je typ prováděné operace. Druhým parametrem se předává ukazatel na strukturu, která obsahuje jednotlivé parametry operace.

Operace `EVTCHNOP_bind_virq` vytvoří nový lokální port, který spojí s virtuálním IRQ. Ukázka 3.4 zobrazuje strukturu definující parametry operace. Každé VIRQ je klasifikováno jako globální nebo vztahující se k virtuálnímu procesoru. Globální VIRQ musí být vždy při alokaci spojeny s VCPU0, avšak je možné je poté spojit s jiným procesorem pomocí operace `EVTCHNOP_bind_vcpu`. Klasifikace jednotlivých VIRQ je uvedena v souboru `xen/include/public/xen.h`.

Ukázka 3.4: Struktura definující parametry operace `EVTCHNOP_bind_virq` (soubor `xen/include/public/xen.h`)

```
struct evtchn_bind_virq {
    /* IN parameters. */
    uint32_t virq;
    uint32_t vcpu;
    /* OUT parameters. */
    evtchn_port_t port;
};
```

Operace `EVTCHNOP_bind_pirq` vytvoří nový lokální port a spojí jej s fyzickým IRQ. Tato operace je uvedena pouze pro úplnost, jelikož není v implementační části použita.

Operace `EVTCHNOP_bind_ipi` vytvoří nový lokální port asociovaný se zadaným VCPU. Tento port poté může být použit pro vyvolání přerušení na daném VCPU. Ukázka 3.5 zobrazuje strukturu definující parametry operace.

Ukázka 3.5: Struktura definující parametry operace `EVTCHNOP_bind_ipi` (soubor `xen/include/public/xen.h`)

```
struct evtchn_bind_ipi {
    uint32_t vcpu;
    /* OUT parameters. */
    evtchn_port_t port;
};
```

Operace `EVTCHNOP_bind_interdomain` vytvoří nový lokální port. Ten je spojen kanálem událostí s portem vzdáleným. Tento kanál lze poté využít pro mezidoménovou komunikaci. Ukázka 3.6 zobrazuje strukturu definující parametry operace.

Ukázka 3.6: Struktura definující parametry operace `EVTCHNOP_bind_interdomain` (soubor `xen/include/public/xen.h`)

```
struct evtchn_bind_interdomain {
    /* IN parameters. */
    domid_t remote_dom;
    evtchn_port_t remote_port;
    /* OUT parameters. */
    evtchn_port_t local_port;
};
```

Operace `EVTCHNOP_alloc_unbound` vytvoří nový lokální port, na který se bude moci vzdálená doména napojit (viz operace `EVTCHNOP_bind_interdomain`). Ukázka 3.7 zobrazuje strukturu definující parametry operace. Výstupním parametrem je položka `port` (nově vytvořený lokální port). Vstupními parametry jsou

`dom`, který určuje, ve které doméně bude port vytvořen a `remote_dom`, ten určuje, která doména se může na port napojit. Pro neprivilegovanou doménu musí být jako `dom` uvedeno `DOMID_SELF`. Rovněž je možné uvést `DOMID_SELF` jako `remote_dom` pro implementaci loopback spojení.

Ukázka 3.7: Struktura definující parametry operace `EVTCHNOP_alloc_unbound` (soubor `xen/include/public/xen.h`)

```
struct evtchn_alloc_unbound {
    /* IN parameters */
    domid_t dom, remote_dom;
    /* OUT parameters */
    evtchn_port_t port;
};
```

Událost lze přes zadaný port generovat operací `EVTCHNOP_send`. Ukázka 3.8 zobrazuje strukturu definující parametry operace.

Ukázka 3.8: Struktura definující parametry operace `EVTCHNOP_send` (soubor `xen/-include/public/xen.h`)

```
struct evtchn_send {
    /* IN parameters. */
    evtchn_port_t port;
};
```

3.7 Práce s časem

Ve virtuálním prostředí lze rozlišit čas skutečný, tedy ten, který "tiká" i ve chvíli, kdy není doména naplánována, a čas virtuální, ten "tiká", jen pokud běží doména. Skutečný čas si lze představit jako čas na nástěnných hodinách (anglicky "wallclock"). Virtuální čas má význam zejména při plánování.

3.7.1 Časové údaje poskytované Xenem

Xen poskytuje několik časových údajů, pomocí nichž lze spočítat rovněž aktuální skutečný čas.

Čas spuštění domény poskytuje skutečný čas, kdy byla doména spuštěna. Tento čas lze zjistit pomocí struktury `shared_info` a jejích položek začínajících na `wc_`. Přefix `wc` je zkratkou za wallclock. Položka `wc_sec` a `wc_nsec` uvádí čas spuštění domény[16]. Položka `wc_version` je inkrementována pokaždé, když je čas aktualizován. Pokud je nastaven nejnižší bit, potom se čas právě aktualizuje a doména by měla se čtením těchto hodnot počkat.

Počet cyklů je přístupný pomocí registru TSC, stejným způsobem jako v nevirtualizovaném prostředí.

Systémový čas poskytuje informaci, kolik skutečného času uplynulo od spuštění domény. Systémový čas je poskytován odděleně pro každý procesor pomocí struktury `vcpu_time_info`. Struktura obsahuje položku `system_time`, která určuje počet nanosekund od startu systému. Další položkou je položka `tsc_timestamp`, která reprezentuje obsah registru TSC při aktualizaci systémového času. Položka `version` je inkrementována pokaždé, když

[16] Počítáno relativně od 1.1.1970

je aktualizován systémový čas. Pokud je nastaven nejnižší bit na jedna, potom je systémový čas právě aktualizován. Systémový čas je Xenem aktualizován pouze občas, pro zpřesnění je použit registr TSC. Pro výpočet lze použít následující vzorec: `((((tsc - tsc_timestamp) « tsc_shift) * tsc_to_system_mul) » 32)`. Při implementaci je třeba dát pozor na fakt, že `tsc_shift` může být záporný, a tudíž je třeba provést obrácený bitový posun.

Aktuální skutečný čas se spočte součtem času spuštění domény a času systémového. Před zahájením výpočtu je třeba uložit kopii verze systémového i času spuštění domény. Zároveň je nutné vyčkat, dokud nebude poslední bit obou verzí nulový. Po provedení výpočtu je nezbytné pomocí uložené kopie verzí ověřit, že se v průběhu výpočtu verze časových údajů nezměnila. Pokud ano, je třeba výpočet opakovat.

Položky `tsc_shift` a `tsc_to_system_mul` slouží k převodu tiků procesoru na jednotku času. Používají se zejména při výpočtu systémového času, avšak lze pomocí nich zjistit také frekvenci procesoru následujícím vzorcem (v jednotkách Hz): `((1000000000 « 32) / tsc_to_system_mul) » tsc_shift`.

3.7.2 Virtuální čas

Virtuální čas je doméně zpřístupněn pomocí virtuálního časovače. Tento časovač je určen především pro účely plánování. Virtuální časovač používá virtuální IRQ, jehož číslo je definováno makrem `VIRQ_TIMER`. Pro doručení přerušení od časovače prostřednictvím kanálu událostí je třeba napojit virtuální IRQ časovače na kanál událostí, viz část věnovaná přerušení. Ve výchozím nastavení bude doručeno přerušení každých 10ms prostřednictvím periodického časovače.

Virtuální časovač lze nastavit pomocí hypervolání `vcpu_op`. Operace `VCPUOP_set_periodic_timer` nastaví periodu doručení přerušení od časovače, operace `VCPUOP_stop_periodic_timer` zastaví periodický časovač. Dále je podporován také jednorázový časovač, který slouží k doručení jedné události v zadaný systémový čas a lze jej nastavit pomocí operace `VCPUOP_set_singleshot_timer` a případně zrušit `VCPUOP_stop_singleshot_timer`. Více informací viz `xen/include/public/vcpu.h`.

3.8 Zařízení

Zařízení je doménám zprostředkováno prostřednictvím split (v překladu rozdělených, avšak budeme uvádět v originálním znění) ovladačů. Split ovladač se skládá ze dvou částí. Backend část, která má přímý přístup k hardwaru a běží typicky v doméně nula. Frontend část, která je součástí domény. Obě tyto části spolu komunikují prostřednictvím sdílené paměti, kanálů událostí a případně databáze Xenstore.

Zjednodušený scénář komunikace frontend a backend části ovladače je následující. Frontend část ovladače vytvoří pomocí grant tabulek sdílenou paměť, inicializuje kruhový buffer, alokuje nový kanál událostí a tyto zveřejní prostřednictvím databáze Xenstore. Backend tyto údaje přečte, vytvoří si lokální port, který spojí kanálem se vzdáleným portem, namapuje sdílenou paměť. V tuto chvíli již může

frontend část zapisovat požadavky do sdíleného kruhového bufferu. Upozornění o nových požadavcích jsou zasílána prostřednictvím kanálu událostí. Backend část poté zapíše odpověď a opět provede upozornění prostřednictvím kanálu událostí.

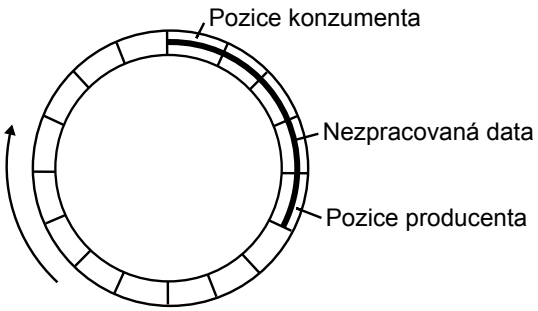
Xen definuje formální protokol, kterým probíhá spojení frontend a backend části ovladače zařízení, které je připojeno k virtuální sběrnici Xenu, které se říká Xenbus. Xenbus je ve skutečnosti pouze databáze Xenstore, nad kterou je definován daný protokol, ten je popsán na [12].

Xen poskytuje podporu pro několik typů zařízení. Jedná se o síťovou kartu, bloková zařízení, grafický framebuffer a samozřejmě konzoli. Krom toho lze umožnit doménám i přímý přístup k hardwaru, avšak tato technika nebude v rámci této práce rozebrána.

3.8.1 Kruhový buffer

Kruhový buffer (anglicky "ring buffer") je v prostředí Xenu velice často využívaný mechanismus pro výměnu informací mezi doménami přes sdílenou paměť. Využívá se zejména pro komunikaci mezi frontend a backend částí ovladače.

Kruhový buffer je souvislá část paměti, nad kterou jsou definovány dva indexy. Jeden index ukazuje pozici producenta, druhý index pozici konzumenta (viz obrázek 3.4). Při zápisu dat se inkrementuje index producenta. Při přečtení dat se inkrementuje index konzumenta. Index do lineární paměti se získá operací modulo.



Obrázek 3.4: Princip kruhového bufferu

Implementace v Xenu

Xen poskytuje podporu pro implementaci vlastního kruhového bufferu. Tato implementace je však vhodná pouze pro případ, kdy každý požadavek má právě jednu odpověď, to je z důvodu, že požadavky a odpovědi využívají pouze jeden kruhový buffer, není tedy jeden pro požadavky a druhý pro odpovědi. Je využit například u ovladače blokových nebo síťových zařízení Xenu. Veškerá makra pro použití tohoto kruhového bufferu a makra pro definici typů jsou definována v souboru `xen/include/public/io/ring.h`.

Pro použití vlastního kruhového bufferu je neprve třeba vytvořit si struktury pro požadavek (např. `request_t`) a odpověď (např. `response_t`). Poté je třeba definovat nový kruhový buffer pomocí makra:

```
DEFINE_RING_TYPES(mytag, request_t, response_t)
```

Toto makro definuje tři typy:

mytag_sring_t Sdílená část kruhového bufferu.
mytag_front_ring_t Privátní část pro frontend.
mytag_back_ring_t Privátní část pro backend.

Před použitím kruhového bufferu je třeba jej inicializovat:

```
mytag_front_ring_t ring;
SHARED_RING_INIT((mytag_sring_t *)shared_page);
FRONT_RING_INIT(&ring, (mytag_sring_t *)shared_page, PAGE_SIZE);
```

Sdílená část kruhového bufferu je inicializována pouze frontend částí. Backend část inicializuje pouze svou privátní část:

```
BACK_RING_INIT(&ring, (mytag_sring_t *)shared_page, PAGE_SIZE);
```

Zápis požadavku (frontend)

Pro zapsání požadavku se použije následující postup. Nejprve získáme další požadavek z kruhového bufferu:

```
RING_IDX i = front_ring.req_prod_pvt;
request_t *req = RING_GET_REQUEST(&front_ring, i);
```

Po vyplnění proměnné `req` je třeba aktualizovat privátní kopii producent indexu.

```
front_ring.req_prod_pvt = i + 1;
```

Tento postup lze opakovat, dokud nebudou zapsány všechny požadavky. Poté se provede operace PUSH a případné upozornění backend části:

```
RING_PUSH_REQUESTS_AND_CHECK_NOTIFY(&front_ring, notify);
if (notify)
    notify();
```

Makro aktualizuje index `req_prod` sdílené části kruhového bufferu indexem `req_prod_pvt` z privátní části kruhového bufferu. Zároveň zjistí, jestli je třeba zasílat upozornění pro backend část. To není občas třeba. Je tomu tak ve chvíli, kdy má backend část ještě nezpracované požadavky, jelikož backend vždy kontroluje nové požadavky po zápisu odpovědi.

Přečtení odpovědi (frontend)

O zapsání odpovědi do kruhového bufferu je frontend část informována příslušným kanálem událostí. Odpověď je z kruhového bufferu přečtena pomocí makra `RING_GET_RESPONSE`. První parametr je adresa privátní části kruhového bufferu a druhý parametr je index (`front_ring.rsp_cons`).

```
response_t *rsp = RING_GET_RESPONSE(&front_ring, cons);
```

Poté je třeba inkrementovat privátní index konzumenta (`front_ring.rsp_cons`). Nakonec by měla frontend část provést následující kontrolu:

```
RING_FINAL_CHECK_FOR_RESPONSES(&front_ring, moretodo);
```

Makro provede zjištění, jestli jsou v kruhovém bufferu ještě nezpracované odpovědi. Pokud ano, bude `moretodo=TRUE`, jinak bude `moretodo=FALSE`. Backend část použití tohoto makra frontend částí předpokládá a snižuje se tak počet zasílaných upozornění. Upozornění není totiž zasíláno, pokud frontend část ještě nezpracovala všechny odpovědi.

Backend

Pro backend část ovladače existují analogická makra (zpravidla stačí request zaměnit za response a opačně), avšak nebudou blíže rozebírána, jelikož nejsou v implementační části využita.

3.8.2 Xenstore

Xenstore je sdílená hierarchická databáze mezi doménami, velmi podobná souborovému systému. Prostřednictvím Xenstoru jsou například doméně poskytnuty informace o zařízení, které jsou dostupné.

Komunikace

S databází Xenstore se komunikuje prostřednictvím sdílené paměti a kanálu událostí. Obě tyto informace lze zjistit ze start info stránky. Rozhraní databáze Xenstore je definováno strukturou `xenstore_domain_interface` (viz ukázka 3.9). Struktura obsahuje dva kruhové buffery, jeden pro dotazy, druhý pro odpovědi, včetně indexů odpovídajícím aktuální pozici v kruhových bufferech.

Ukázka 3.9: Struktura definující rozhraní k databázi Xenstore (soubor `xen/include/public/io/xs_wire.h`)

```
struct xenstore_domain_interface {
    char req[XENSTORE_RING_SIZE]; /* Requests to xenstore daemon. */
    char rsp[XENSTORE_RING_SIZE]; /* Replies and async watch events. */
    XENSTORE_RING_IDX req_cons, req_prod;
    XENSTORE_RING_IDX rsp_cons, rsp_prod;
};
```

Komunikace probíhá asynchnonně. Zjednodušený scénář komunikace s databází Xenstore probíhá nasledovně:

1. Do kruhového bufferu požadavků je zapsána zpráva reprezentující požadavek. Ke zprávě je připojeno unikátní ID, aby bylo možné spojit odpověď s požadavkem.
2. Je provedena notifikace prostřednictvím kanálu událostí.
3. Xenstore požadavek zpracuje a zapíše odpověď do kruhového bufferu odpovědí a zašle notifikaci prostřednictvím kanálu událostí.
4. Odpověď je přečtena a zpracována.

Požadavek i odpověď začíná vždy hlavičkou definovanou strukturou `xsd_sockmsg` (viz ukázka 3.10) následovanou samotnými daty. V hlavičce jsou uvedeny všechny důležité informace, jako je typ, identifikátor požadavku, případné id transakce a délka dat.

Ukázka 3.10: Struktura definující hlavičku požadavku (soubor `xen/include/public/io/xs_wire.h`)

```
struct xsd_sockmsg
{
    uint32_t type;  /* XS_??? */
    uint32_t req_id;/* Request identifier, echoed in daemon's response.  */
    uint32_t tx_id; /* Transaction id (0 if not related to a transaction). */
    uint32_t len;   /* Length of data following this. */

    /* Generally followed by nul-terminated string(s). */
};
```

Struktura

Xenstore je hierarchická databáze do jisté míry podobná souborovému systému. Každý klíč v databázi má hodnotu a případně další podklíče. Hodnota klíče je řetězec, žádné jiné typy nejsou podporovány. Na rozdíl od souborového systému může mít klíč hodnotu i podklíče. Jelikož je struktura databáze podobná souborovému systému, budeme nazývat klíče, které mají podklíče, adresáři.

Adresář `/vm/<uuid>` obsahuje základní konfiguraci domény. Virtuální stroj je identifikován pomocí uuid. Ten se nemění ani při restartu nebo migraci domény na jiný stroj, na rozdíl od domId, který identifikuje konkrétní instanci virtuálního stroje. Obsah adresáře je následující:

uuid uuid domény, redundandní informace, jelikož je již součástí cesty.

name Název virtuálního stroje, informace uvedena v konfiguračním souboru.

memory Velikost paměti, která je pro doménu vyhrazena v megabytech.

vcpus Počet vyhrazených procesorů pro doménu.

vcpu_avail Bitová mapa procesorů, které může doména použít.

image Adresář, který obsahuje základní informace o jádru hostujícího operačního systému.

- **ostype** Typ domain builderu (linux nebo VMX).
- **kernel** Cesta k jádru domény v rámci domény 0.
- **cmdline** Příkazová řádka, která je předána jádru (viz 3.3.1).
- **ramdisk** Cesta k ramdisku, který je předán doméně (viz 3.3.1).

on_reboot Akce provedená při restartu domény (destroy nebo reboot).

on_crash Akce provedená při pádu domény (destroy nebo reboot).

on_poweroff Akce provedená při restartu domény (destroy nebo reboot).

Pro každou doménu rovněž existuje složka `/local/domain/<domainId>`, jejíž obsah je následující:

domId Identifikátor domény

name viz `/vm/<uuid>`

store Adresář obsahující základní informace pro komunikaci s databází Xenstore. Informace jsou rovněž uvedeny ve `start_info` struktuře.

- **ring-ref** Grant reference sdílené paměti sloužící ke komunikaci backend a frontend části ovladače Xenstore.

port Lokální port určený pro zasílání/přijímání upozornění v případě nových požadavků/odpovědí v kruhovém bufferu databáze Xenstore.

vm Cesta ve tvaru `/vm/<uuid>`

image Adresář obsahující parametry jádra.

entry viz 3.2.2

pae-mode viz 3.2.2

paddr-offset viz 3.2.2

virt-base viz 3.2.2

hypercall-page viz 3.2.2

guest-os viz 3.2.2

xen-version viz 3.2.2

guest-version viz 3.2.2

loader viz 3.2.2

console Adresář obsahující informace o konzoli.

cpu Adresář obsahující informace o daném virtuálním procesoru.

3.8.3 Bloková zařízení

Bloková zařízení jsou zpřístupněna pomocí klasického split ovladače a jsou nastavena v konfiguračním souboru domény v doméně nula. Jako blokové zařízení lze použít například klasický soubor na disku, fyzický disk nebo svazek LVM, avšak ke všem se z frontend části přistupuje naprosto stejně.

Informace, která bloková zařízení má doména k dispozici, je přístupná prostřednictvím databáze Xenstore.

Komunikace

Komunikace mezi frontend a backend částí ovladače probíhá standardně prostřednictvím kruhového bufferu a kanálu událostí. Pro požadavek a odpověď jsou definovány struktury[17] `blkif_request`, resp. `blkif_response`. Tyto struktury slouží k předávání informací pomocí kruhového bufferu, viz 3.8.1. Položky struktury požadavku jsou následující:

id Identifikátor, který bude použit v odpovědi. Umožňuje spojit požadavek s odpovědí.

operation Požadovaná operace, uvede se buď čtení (`BLKIF_OP_READ`) nebo zápis (`BLKIF_OP_WRITE`).

sector_number Číslo sektoru, od kterého se začne číst/zapisovat.

seg[] Pole jednotlivých datových segmentů. Segment je definován strukturou `blkif_request_segment`. U každého segmentu je třeba vyplnit grant referenci (sdílenou stránku) a číslo prvního a posledního sektoru. Při velikosti stránky 4KB a sektoru 512KB[18] lze do jedné stránky uložit až 8 sektorů. Jeden segment tedy nemůže být větší než 4KB. Číslo prvního sektoru určuje

[17] Soubor `xen/include/public/io/blkif.h`

[18] V současnosti jediná možná volba

číslo sektoru na disku relativně od `sector_number`, číslo posledního sektoru zase číslo posledního.

nr_segments Počet datových segmentů.

handle Identifikátor zařízení, nad kterým je operace prováděna. Jedná se o číselnou hodnotu poslední části cesty k zařízení v databázi Xenstore.

Struktura odpovědi obsahuje identifikátor, který byl uveden v požadavku, operaci, která byla uvedena v požadavku a výsledek `BLKIF_RSP_???`.

Aby spolu mohly obě části ovladače komunikovat, musejí obě strany přejít do stavu `connected`. Na začátku se obě strany nacházejí ve stavu `initializing`. Přechod je zahájen frontend částí ovladače. Ta nejdříve vytvoří sdílenou paměť a alokuje lokální port pro mezidoménovou komunikaci. Tyto údaje zapíše do databáze Xenstore a poté přejde do stavu `initialized`. Backend část ovladače sleduje stav frontend části a zareaguje tak, že zapíše do databáze Xenstore údaje o geometrii disku, připojí se na kanál událostí, namapuje sdílenou paměť do svého prostoru (ne nutně v tomto pořadí) a přejde do stavu `connected`. Poté rovněž frontend část změní stav na `connected` a obě části spolu mohou začít komunikovat. Jednotlivé stavy a přechody mezi stavy jsou definovány protokolem Xenbus.

Popis adresářů v databázi Xenstore

V databázi Xenstore je vyhrazen adresář jak pro frontend, tak pro backend část ovladače. Adresář frontend části je `/local/domain/<domId>/device/vbd/<virtualDevice>/`, jekož obsah je následující:

virtual-device Číslo frontend části ovladače.

backend-id Cesta k backend části ovladače v databázi Xenstore.

backend Adresář obsahující základní informace o konzoli.

ring-ref Grant reference sdílené paměti určené pro komunikaci nezi frontend a backend částí ovladače.

event-channel Kanál událostí pro komunikaci backend a frontend části ovladače.

Ovladač frontend části získá adresář backend části pomocí klíče `backend` a není tedy důležité, kde se konkrétní adresář nachází. Adresář backend části obsahuje klíče:

frontend-id Id domény, ve které se nachází frontend část ovladače.

state Stav backend části ovladače.

frontend Cesta k frontend části ovladače v databázi Xenstore.

sector-size Velikost sektoru v bytech (zapsáno backendem až ve stavu `initialized`).

sectors Počet sektorů (zapsáno backendem až ve stavu `initialized`).

info Dodatečné informace spojené se zařízením. Informace jsou poskytnuty pomocí několika flagů, které jsou spojeny operací OR. Jedná se o `CDROM=1`, `REMOVABLE=2`, `READ_ONLY=4`.

params Parametry zařízení. Obsahuje cestu k zařízení v rámci Dom0.

type Typ backendu. Možné jsou `phy` v případě, že backend je fyzický disk (popř. LVM disk) nebo `file` v případě, že disk je soubor.

dev Název zařízení v rámci vytvořené domény. Uvedeno v konfiguračním souboru.

3.8.4 Konzole

Přístup ke konzoli je zprostředkován pomocí sdílené paměti a kanálu událostí. Sdílená stránka i příslušný kanál jsou přístupné přímo ze start info stránky, to je zejména z toho důvodu, že se předpokládá, že konzole bude třeba již v raných fázích inicializace hostujícího operačního systému.

Rozhraní je definováno strukturou `xencons_interface` (viz ukázka 3.11). Tato struktura definuje dva kruhové buffery. Jeden je určený pro vstup a druhý pro výstup. Pro každý kruhový buffer definuje struktura dva indexy a to je pozice producenta (`in_prod`, resp. `out_prod`) a pozice konzumenta (`in_cons`, resp. `out_cons`).

Ukázka 3.11: Struktura definující rozhraní konzole (soubor `xen/include/public/io/console.h`)

```
struct xencons_interface {
    char in[1024];
    char out[2048];
    XENCONS_RING_IDX in_cons, in_prod;
    XENCONS_RING_IDX out_cons, out_prod;
};
```

Vstup

Pokud je proveden vstup, zapíše backend část ovladače do kruhového bufferu příslušná data, aktualizuje index `in_prod` a upozorní frontend část prostřednictvím kanálu událostí. Frontend část přečte data a aktualizuje příslušný index `in_cons`.

Výstup

Výstup probíhá obdobným způsobem jako vstup. Do kruhového bufferu jsou zapsána data, aktualizován index `out_prod` a je upozorněna backend část ovladače, která data přečte a zpracuje, přičemž aktualizuje index `out_cons`.

Formát dat

Data, která jsou konzolí přenášena, jsou definována protokolem textového terminálu, ke kterému je konzole připojena. Typicky se jedná o VT100 kompatibilní terminál.

3.8.5 Ladicí vstup/výstup

Pokud je Xen překompilován s podporou ladění, je možné využít ladicího vstupu a výstupu. Tyto jsou zprostředkovány prostřednictvím hypervolání `console_io(int cmd, int count, char *str)`. Prvním parametrem je operace. První možnou operací je `CONSOLEIO_write`, která zapíše `count` znaků z bufferu `str`

na výstup. Druhou možnou operací je `CONSOLEIO_read`, která přečte maximálně `count` znaků ze vstupu do bufferu `str`. Vrací počet přečtených znaků.

3.8.6 Další zařízení

Další zařízení budou pouze zmíněna, avšak nebudou podrobně rozebírána zejména proto, že nejsou součástí implementační části této práce.

Síťová karta

Virtuální síťová karta využívá opět modelu split ovladače. V principu probíhá komunikace s backend částí stejným způsobem jako v případě blokových zažízení. Více informací lze získat např. v dokumentaci rozhraní Xenu [6].

Grafická konzole

Xen poskytuje rovněž přístup ke grafické konzoli. Výstup probíhá pomocí virtuálního framebufferu a vstup prostřednictvím virtuální klávesnice a polohovacího zařízení. Z uživatelského pohledu je konzole přístupná prostřednictvím VNC serveru. Více informací lze získat v [5].

4. Popis systému HelenOS

Cílem této kapitoly je popis základní struktury systému HelenOS a částí, které jsou nezbytné pro pochopení dalšího textu a orientaci ve zdrojových kódech prototypové implementace, nikoli kompletní dokumentace. Tu lze nalést na stránkách systému HelenOS[1].

HelenOS je mikrokernelový operační systém, což znamená, že v privilegovaném režimu jádra je umístěna pouze nezbytně nutná část funkcionality celého operačního systému. Zbytek se přesouvá do uživatelského prostoru, kde je zbylá funkcionalita implementována řadou uživatelských úloh[2]. Jádro poskytuje úlohám možnost mezi sebou komunikovat prostřednictvím meziprocesové komunikace (IPC).

V současné době obsahuje HelenOS port na 7 architektur, konkrétně AMD64/-EM64T (x86-64), ARM, IA-32, IA-64 (Itanium), 32-bit MIPS, 32-bit PowerPC a SPARC V. Mnohé z nich byly implementovány v rámci jiných diplomových prací.

4.1 Struktura jádra

Jádro systému HelenOS striktně odděluje část architektonicky závislou od části generické. Zdrojové kódy jádra jsou rozděleny do tří složek:

arch/ Obsahuje části systému HelenOS, které jsou závislé na architektuře. Jedná se například o kód obstarávající zavedení systému a implementaci rozhraní architektury.

genarch/ Jedná se o části, které jsou společné pro více architektur. Například generická implementace stránkovacích tabulek, kterou lze použít pro více architektur.

generic/ Platformě nezávislá část jádra.

Z funkčního hlediska lze jádro rozdělit na několik subsystémů, které dohromady tvoří celkovou funkčnost. Subsystémy odpovídají zhruba podadresářům `generic/src/`, popř. `arch/`*název architektury*`/src/`. Kompletní seznam a funkce subsystémů lze nalést v [3].

4.2 Meziprocesová komunikace

HelenOS poskytuje IPC ve formě zasílání zpráv. Zasílání zpráv funguje na analogii telefonního hovoru. Pokud se chce úloha spojit s jinou, musí znát její telefonní číslo, poté použije svůj telefon, zavolá a nechá vzkaz na záznamníku. Druhá strana si přečte vzkaz ze záznamníku a zavolá protistraně na její záznamník odpověď.

Podpora pro meziprocesovou komunikaci je poskytována jádrem operačního systému. Rozhraní pro zasílání zpráv je zpřístupněno prostřednictvím systémových volání. Používání tohoto rozhraní je však pro procesy příliš komplikované.

[1] http://www.helenos.org

[2] V systému HelenOS jsou procesy nazývány úlohami (anglicky "task"), což je obvyklá terminologie zejména mikrokernelových operačních systémů

Z tohoto důvodu vznikl v uživatelském protoru tzv. asynchronní framework, který v co největší míře usnadňuje meziprocesovou komunikaci. Více informací o meziprocesové komunikaci v systému HelenOS lze zjistit na [8].

4.3 Správa paměti

Správa paměti je zajištěna kompletně jádrem a implementována subsystémem `mm`. Jádro má na starosti správu fyzické paměti i virtuálních adresových prostorů jednotlivých úloh.

4.3.1 Fyzická paměť

Fyzická paměť je rozdělena na tzv. zóny. Každá zóna obsahuje informace o svém umístění a velikosti, informaci o volných rámcích prostřednictvím Buddy alokátoru a flagy, které určují, jaký typ přístupu zóna umožňuje (pouze pro čtení/pro zápis).

4.3.2 Virtuální adresový prostor

Adresový prostor procesu je rozdělen na oblasti (anglický originál "area"). Oblast je část paměti, která má stejné vlastnosti a stejný způsob mapování. Každá oblast má svůj backend, který určuje mimo jiné, jakým způsobem jsou řešeny výpadky stránek. V současné implementaci poskytuje systém HelenOS několik backendů:

elf Slouží pro mapování obsahu souboru v ELF formátu do operační paměti.

anon Anonymní oblast paměti nemapuje přímo žádná data. Při výpadku stránky je alokován nový rámec, který je namapován.

phys Slouží k namapování souvislé části fyzické paměti.

V rámci prototypové implementace je přidán nový backend určený pro sdílenou paměť mezi doménami. Více bude popsán v kapitole věnované implementaci.

4.4 Ovladače

HelenOS je mikrokernelový operační systém a jako takový přesouvá co nejvíce ovladačů do uživatelského prostoru. Aby bylo možné ovladače v uživatelském prostoru implementovat, poskytuje jádro rozhraní, pomocí něhož lze k hardwaru přistupovat z uživatelského prostoru a zpracovávat přerušení.

4.4.1 Přístup k fyzické paměti

Jádro systému nabízí možnosti namapovat část fyzické paměti do uživatelského prostoru. Namapování je provedeno prostřednictvím systémového volání `physmem_map`. Toto umožňuje například přístup k paměťově mapovaným registrům. Není však možné namapovat libovolnou část fyzické paměti. Aby bylo možné namapovat fyzickou paměť, musí být příslušná oblast fyzické paměti registrována (mapování je rovněž povoleno, pokud není mapovaná paměť součástí žádné zóny

nebo je součástí zóny označené jako firmware). Registrace oblasti je provedena z jádra systému funkcí `ddi_parea_register`. Systémové volání `physmem_map` ověří oprávnění a vytvoří novou oblast virtuální paměti s backendem `backend_phys`. Vložení záznamu do stránkovacích tabulek je provedeno až v rámci zpracování výpadku stránky.

4.4.2 Zpracování IRQ

Nedílnou součástí ovladače zařízení je zpracování přerušení. HelenOS obsahuje framework pro zpracování přerušení v uživatelském prostoru. Registrace IRQ je provedena zavoláním funkce `ipc_register_irq`. Pomocí pseudokódu lze specifikovat akce, které má provést samotné jádro po příchodu přerušení a umožňuje například provést čtení nebo zápis registru zařízení, přijmout nebo zamítout přerušení aj. Dále je třeba zaregistrovat handler přerušení prostřednictvím funkce `async_set_interrupt_received`, tento handler bude vyvolán při příchodu přerušení a ovladač jej bude moci odpovídajícím způsobem zpracovat.

4.4.3 Registry zařízení

Systém HelenOS obsahuje podporu pro přístup k registrům zařízení z uživatelského prostoru. Jelikož ale přístup ke skutečným zařízením není součástí práce, je tato možnost uvedena pouze pro úplnost a nebude dále rozebírána.

4.4.4 Framework pro ovladače

Pro podporu ovladačů v uživatelském prostoru byl vytvořen framework. Tento framework umožňuje automatickou detekci zařízení a zavedení odpovídajícího ovladače. Více informací o frameworku pro ovladače viz [9].

4.4.5 Sysinfo

Sysinfo je databáze typu klíč-hodnota implementovaná v jádře operačního systému zpřístupněná do uživatelského protoru prostřednictvím systémových volání. Slouží zejména pro předávání informací z jádra systému do uživatelského protoru.

4.5 Plánování

Jádro systému HelenOS plánuje preemtivně na úrovni vláken. Vždy, když přijde přerušení od časovače, je zavolán plánovač (funkce `schedule`), který provede přeplánování. Plánovač nedělá žádné rozdíly mezi vlákny jádra a uživatelského prostoru.

4.5.1 Vlákna v uživatelském prostoru

V uživatelském prostoru se dále vlákno dělí na fibrily (vlákna implementovaná v uživatelském prostoru). Fibrily jsou spravovány a kooperativně plánovány prostřednictvím libc. Každý fibril má svůj kontext a jádro nemá o jejich existenci

tušení. K přeplánování fibrilu dojde v několika možných případech. Jedná se například o situaci, kdy se čeká v rámci synchronizace nebo když fibril explicitně zavolá operaci yield.

4.6 Zdrojové kódy a kompilace

HelenOS využívá pro správu verzí Bazaar repository. Hlavní vývojovou linii lze stáhnout příkazem:

```
bzr branch bzr://bzr.helenos.org/mainline HelenOS
```

Kompilace se spustí příkazem `make` z kořenové složky zdrojových kódů. Pokud je systém kompilován poprvé, je spuštěna konfigurace, která umožňuje systém nastavit dle potřeb. Tuto konfiguraci lze vyvolat také později příkazem `make config`. Více informací o kompilaci viz [9].

5. Analýza

Cílem této kapitoly je analyzovat z vyššího pohledu změny, které bude třeba provést v systému HelenOS pro funkční implementaci paravirtualizace. Rozebrány budou nejen vlastnosti, které implementovány budou, ale také vlastnosti, které zejména z časových důvodů implementovány nebudou, avšak nejsou pro běh paravirtualizovaného operačního systému nezbytné.

Implementaci paravirtualizace lze rozdělit na dvě části. První částí je port jádra operačního systému. Port jádra zahrnuje implementaci změn v jednotlivých subsystémech a rané inicializaci jádra. Druhou částí je port uživatelského prostoru. Ten zahrnuje zejména implementaci nových ovladačů a port stávajích ovladačů. Aby mohly být implementovány paravirtualizované ovladače v uživatelském prostoru, bude dále nutné zpřístupnit část rozhraní Xenu do uživatelského prostoru.

5.1 Změny v jádře

Změnou úrovně oprávnění nemůže jádro provádět určité privilegované operace nativně. Tyto operace jsou zpřístupněny voláním služeb hypervisoru, ty jsou popsány v kapitole 3. Jelikož se rozhraní hypervisoru snaží do značné míry kopírovat rozhraní nativní, není nutné vytvářet port úplně od začátku, avšak lze vyjít ze stávajícího portu IA-32.

Změny v jádře systému jsou očekávány téměř výhradně v architektonicky specifické části kódu. Zřejmě první nutná úprava se bude týkat samotného způsobu zavedení jádra pod hypervisorem Xen, které se v paravirtualizovaném prostředí značně liší. Poté bude třeba naportovat jednotlivé subsystémy.

Změny se týkají subsystémů, které mají architektonicky specifickou část. Tato architektonicky specifická část je implementována v adresáři arch. Jedná se typicky o několik funkcí, které jsou využity při implementaci subsystému v generické části. Souhrn těchto rozhraní lze označit za rozhraní architektury.

Existuje několik možných přístupů při implementaci paravirtualizace do jádra operačního systému. První možností je kompletně implementovat rozhraní architektury stejným způsobem, jako je tomu u jiných podporovaných architektur.

Dalším možným postupem je vytvoření nového rozhraní, které bude více kopírovat jednotlivé funkce ovlivněné paravirtualizací a to bude použito pro úpravu stávajícího portu IA-32. Toto rozhraní lze pak implementovat pro nativní variantu, tak pro paravirtualizovanou.

Paravirtualizační vrstva má nespornou výhodu v tom, že je možné ji implementovat pro několik paravirtualizérů zároveň, přičemž při vhodné implementaci lze jedno sestavené jádro spustit jak nativně, tak pravirtualizovaně. Tímto způsobem je implementována podpora paravirtualizace například v systému Linux pod názvem PVOPS.

Jinou, zřejmě nejpřímnější metodou je úprava stávajícího portu IA-32 a části ovlivněné paravirtualizací budou nahrazeny již před kompilací prostřednictvím maker. Toto řešení by však znamenalo příliš velké zanesení a znepřehlednění kódu.

Hlavním kritériem pro výběr implementace je, do jaké míry vyhovuje stávající architektonické rozhraní požadavkům paravirtualizace. Toto rozhraní je velice

přímé a přidání další vrstvy pro paravirtualizaci se nezdá opodstatněné.

5.1.1 Stránkovací tabulky

V části věnované rozhraní Xenu jsou uvedeny dvě možné metody, kterými Xen validuje zápisy do stránkovacích tabulek. Jedná se o přímo zapisovatelné stránkovací tabulky a plně paravirtualizovanou variantu, při které se pro změnu musí explicitně zavolat služba hypervisoru.

Každá z těchto variant má své klady a zápory. Přímo zapisovatelné stránkovací tabulky mají nespornou výhodu v tom, že je vytvářena iluze toho, že lze do stránkovacích tabulek přímo zapisovat, což ve svém důsledku může znamenat menší portovací úsilí. Neznamená to však, že stránkovací tabulky budou fungovat stejným způsobem, jako v nativním prostředí. Zapisované adresy je totiž stále třeba převést na machine adresy.

Jelikož jsou však přímo zapisovatelné stránkovací tabulky implementovány prostřednictvím emulace instrukcí, je zřejmě plně paravirtualizovaná varianta rychlejší.

5.1.2 Omezení

Návrh správy paměti operačního systému HelenOS neumožnuje adresovat více než 2GB fyzické paměti. HelenOS totiž identicky mapuje dostupnou fyzickou paměť do horních 2GB paměti. K fyziké paměti je pak přistupováno pouhým přičtením 2GB k fyzické adrese. Tohoto principu je využito u všech podporovaných architektur. Xen se mapuje do hormích 168MB paměti, čímž se adresovatelná fyzická paměť ještě zmenšuje.

Toto omezení se však stává značně limitující v případě implementace PAE. Bez PAE umožňuje procesor adresovat pouze 4GB fyzické paměti (32-bit adresy). Se zapnutým PAE je však možné používat 36-bit fyzické adresy, tedy lze adresovat až 64GB fyzické paměti.

Pro starší verze Xenu bylo možné podporu PAE vypnout. Nové verze však podporu PAE vyžadují. Možným řešením by bylo například vytvořit port pro starší verzi Xenu, avšak toto řešení bylo z pochopitelných důvodů zavrhnuto.

Další možností je podporu PAE implementovat, avšak se zachovaným omezením, jelikož toto omezení existovalo již před implementací paravirtualizace. Implementace plné podpory PAE je mimo rozsah této práce a vyžadovalo by změnu způsobu přístupu k paměti, zejména vytvoření mapování on-demand. Podpora PAE se zachovaným omezením neznamená pro systém větší úpravy. Virtuální adresy zůstávají stále 32-bit. Jedinou úpravu vyžadují stránkovací tabulky, které převádějí 32-bit virtuální adresu na 36-bit fyzickou.

5.2 Ovladače

HelenOS je mikrokernelový operační systém, což mimo jiné znamená, že jsou ovladače, které nejsou nezbytně nutné pro chod jádra, implementovány v uživatelském protoru.

Aby bylo možné implementovat ovladače v uživatelském prostoru, bude třeba vytvořit odpovídající rozhraní ovladačů, které bude umožňovat komunikaci

frontend a backend části. Jelikož port zahrnuje pouze podporu neprivilegované domény, budou rozebrány pouze požadavky frontend části ovladače.

5.2.1 Požadavky ovladače

Ovladače virtuálních zařízení v prostředí Xenu mají dvě části, frontend a backend. Frontend část ovladače komunikuje s backend částí prostředníctvím sdílené paměti, kanálů událostí a databáze Xenstore. Cílem této části je analyzovat požadavky frontend části ovladače, aby bylo možné navrhnout rozhraní umožňující implementaci frontend ovladače v uživatelském prostoru.

Před zahájením spojení se obě části ovladače nachází ve stavu `initializing`. Frontend část ovladače zveřejňuje grant referenci sdílené části paměti a číslo alokovaného kanálu událostí pomocí databáze Xenstore. Ve chvíli, kdy přejde frontend část do stavu `initialized`, zapíše backend část případné detaily o zařízení do databáze Xenstore a přejde do stavu `connected`, poté přechází i frontend část do stavu `connected`.

Aby byla možná komunikace, potřebuje rovněž frontend část zasílat upozornění prostřednictvím alokovaného kanálu a naopak, musí být schopna přijímat upozornění zaslaná backend částí ovladače.

Obecné požadavky frontend části ovladače lze tedy shrnout takto:

Sdílená paměť Frontend část ovladače musí mít možnost vytvořit sdílenou paměť a určit, která doména k ní smí přistupovat a s jakým oprávněním. Pro komunikaci prostřednictvím sdílené paměti stačí sdílení s velikostí jedné stránky. Tato paměť musí být přístupna z virtuálního adresového prostoru ovladače.

Události Frontend část ovladače musí mít možnost vytvořit nový kanál událostí. Přes tento kanál musí být schopna události jednak zasílat, ale také přijímat.

Xenstore Frontend část ovladače musí mít přístup k databázi Xenstore.

Lze předpokládat, že pokud budou splněny všechny tyto požadavky, bude možné implementovat libovolný frontend ovladač, který dodržuje architekturu split ovladačů Xenu.

Bude tedy třeba implementovat rozhraní, umožňující mezidoménové sdílení paměti, rozhraní umožňující alokaci nového kanálu událostí a implementovat mechanismus doručování událostí do uživatelského prostoru. Rovněž je třeba implementovat přístup k databázi Xenstore.

5.2.2 Rozhraní pro ovladače

Pokud by byl ovladač umístěn v jádru operačního systému, odpadly by všechny problémy. Bylo by totiž možné použít přímo hypercall rozhraní hypervisoru. Toto rozhraní však nelze používat z uživatelského prostoru. Pokud ale chceme implementovat ovladače v uživatelském prostoru, bude třeba alespoň část rozhraní zpřístupnit.

První možností se nabízí vytvoření systémového volání "proveď hypercall". Toto systémové volání by bylo velmi jednoduché a pouze by předávalo požadavky hypervisoru. Je třeba si však uvědomit, že v tomto případě by proces v uživateském prostoru získal kompletní kontrolu nad samotným jádrem operačního

systému a všemi prostředky, ke kterým lze přes hypercall rozhraní přistupovat. Dalším problémem by byla špatná kontrola využití zdrojů. Ve chvíli, kdy by si proces prostřednictvím tohoto rozhraní naalokoval různé prostředky, například kanály událostí, grant reference apod., bylo by velmi komplikované dopátrat, kterému procesu patří, a při pádu procesu by nebyly nikdy odalokovány.

Jinou možností je implementovat rozhraní pomocí služeb jádra jako systémová volání. Pro toto řešení mluví zejména fakt, že obdobným způsobem je již řešeno stávající rozhraní pro ovladače. Tato metoda umožňuje rovněž asociovat prostředky s danou úlohou a v případě její ukončení všechny prostředky odalokovat.

Co se týče doručování událostí do uživatelského prostoru, je cílem implementace využít v co největší míře stávajícího mechanismu zpracování přerušení v uživatelském prostoru místo implementace nové infrastruktury. Toto bude provedeno novým systémovým voláním, které umožní napojení kanálu událostí na IRQ.

5.2.3 Databáze Xenstore

Rozhraní k databázi Xenstore je možné implementovat buď v jádře operačního systému, nebo v uživatelském prostoru. Pokud by bylo rozhraní implementováno v jádře, bylo by nezbytné zpřístupnění rozhraní rovněž do uživatelského prostoru (pro frontend ovladače). Jádro operačního systému však databázi Xenstore nikterak nevyužije a je možné jej implementovat čistě v uživatelském prostoru.

V uživatelském prostoru existuje framework pro ovladače, a jelikož je Xenstore ve skutečnosti chápán jako virtuální sběrnice, dávalo by smysl jej implementovat pomocí tohoto frameworku. Tento framework je však relativně nový a současné ovladače blokových zařízení nejsou pomocí tohoto frameworku implementovány. Jelikož je součástí práce rovněž ovladač blokových zařízení, není tento framework využit. Databáze Xenstore bude implementována jako samostatná úloha, se kterou se komunikuje prostřednictvím IPC. Dále bude implementována knihovna, která bude zaobalovat volání IPC metod.

Terminologie: Občas je používán rovněž termín Xenbus místo Xenstore. Tento termín označuje Xenstore jako virtuální sběrnici, ke které jsou připojeny virtuální zařízení. Tyto termíny lze používat téměř zaměnitelně. Při implementaci do operačního systému HelenOS bude Xenstore chápán skutečně jako virtuální sběrnice, proto se budeme v rámci implementace držet termínu Xenbus.

5.2.4 Ovladač blokových zařízení

Pro ověření správnosti rozhraní pro paravirtualizované ovladače bude implementován ovladač blokových zařízení. Původním záměrem bylo využít frameworku pro ovladače. Stávající ovladače blokových zařízení však nejsou dosud pomocí tohoto frameworku implementovány. Ovladač blokových zařízení Xenu bude tedy rovněž implementován stejným způsobem (jako samostatná úloha).

5.3 Vlastnosti, které implementovány nebudou

Ne všechny vlastnosti lze zejména z časových důvodů implementovat. Implementace by však měla obsahovat všechny vlastnosti, které jsou nutné pro funkční běh

operačního systému HelenOS v paravirtulizovaném prostředí. V rámci této části budou rozebrány vlastnosti, které implementovány nebudou.

5.3.1 Virtuální framebuffer

Podpora virtuálního framebufferu pro paravirtualizované domény byla oficiálně součástí Xenu 3.0.4. Ke framebufferu je přistupováno prostřednictvím VNC serveru. Ruku v ruce jde s podporou virtuálního framebufferu i podpora virtuální klávesnice a polohovacího zařízení – myši.

Už samotný fakt, že podpora virtuálního framebufferu byla do Xenu přidána takto pozdě, svědčí o tom, že se nejedná o vlastnost, bez které se nelze obejít, ačkoli je z uživatelského pohledu velmi atraktivní a zřejmě se jedná o metodu interakce s virtuálním strojem, kterou by běžný uživatel očekával.

Neprivilegovaná doména si však plně vystačí s textovou konzolí, která je obdobou komunikace prostřednictvím sériové linky. Textová komunikace probíhá typicky terminálovým protokolem VT100, ten však není nutné plně implementovat pro konzoli jádra, ta slouží zejména k zadávání ladících příkazů a výpisu bootovacích zpráv jádra.

5.3.2 Síťová karta

Podpora virtuální síťové karty není k funkční paravirtualizaci nezbytná a nebude implementována zejména z časových důvodů.

Síťová karta pracuje obdobným způsobem jako blokové zařízení. Pokud by byl v budoucnu implementován ovladač virtuální síťové karty, byl by implementován obdobně jako ovladač blokových zařízení. Při příchodu přerušení by byl přečten packet a předán operačnímu systému ke zpracování. Pokud by naopak byl packet operačním systémem posílán, byl by zapsán požadavek do kruhového bufferu a odesláno upozornění backend části ovladače.

6. Implementace

Tato kapitola popisuje prototypovou implementaci paravirtualizace v systému HelenOS a měla by shrnovat všechny kroky, které byly při implementaci provedeny.

6.1 Rozhraní hypervizoru

Aby bylo možné používat hypervolání z jazyka C, ve kterém je systém HelenOS napsán, byly do systému HelenOS začleněny:

- Veřejné hlavičkové soubory Xenu pro jazyk C. Ty obsahují veškeré datové typy rozhraní a užitečná makra.
- Wrappery hypervolání pro jazyk C.

Veřejné hlavičkové soubory Xenu jsou umístěny v adresáři `abi/include/xen/` a jsou zkopírovány přímo ze zdrojových souborů Xenu. Wrappery hypervolání pro jazyk C bylo možné napsat od základu nové nebo je převzít z jiného operačního systému, který je na hypervisor Xen již naportován. Zvoleným řešením bylo převzetí wrapperů (s drobnými úpravami) z operačního systému MiniOS, který je součástí zdrojových kódů Xenu. Toto rozhraní je umístěno v souboru `kernel/ia32xen/include/hypercall-x86_32.h`.

6.1.1 Wrappery hypervolání pro jazyk C

Hypervolání jsou zpřístupněna pomocí klasického volání funkcí jazyka C. Pro každé hypervolání je vytvořena funkce s prototypem `HYPERVISOR_název_hypervolani(parametr1, ..., parametrN)`, např. pro `mmuext_op` vypadá funkce následovně:

```
static inline int
HYPERVISOR_mmuext_op(
    struct mmuext_op *op, int count, int *success_count, domid_t domid)
{
    return _hypercall4(int, mmuext_op, op, count, success_count, domid);
}
```

Pro každý možný počet parametrů hypervolání je vytvořeno makro, které zaobaluje nízkoúrovňové volání v Assembleru. Pro počet parametrů 4 vypadá makro následovně:

```
#define _hypercall4(type, name, a1, a2, a3, a4)          \
({                                                       \
    long __res, __ign1, __ign2, __ign3, __ign4;          \
    asm volatile (                                       \
        "call hypercall_page + ("STR(__HYPERVISOR_##name)" * 32)"\
        : "=a" (__res), "=b" (__ign1), "=c" (__ign2),    \
        "=d" (__ign3), "=S" (__ign4)                     \
        : "1" ((long)(a1)), "2" ((long)(a2)),            \
        "3" ((long)(a3)), "4" ((long)(a4))               \
        : "memory" );                                    \
    (type)__res;                                         \
})
```

Pro jiný počet parametrů je makro vytvořeno analogicky. Tato makra zobrazují způsob hypervolání prostřednictvím stránky hypervolání.

6.2 Architektura ia32xen

Nový port jádra vychází ze stávajícího portu na architekturu IA-32. V jádru byla vytvořena nová architektura ia32xen (`kernel/arch/ia32xen/`), která vychází ze stávajícího portu na architekturu IA-32 (`kernel/arch/ia32/`).

6.3 Formát a zavedení jádra

Výstupem kompilace systému HelenOS (pro architekturu IA-32) je ISO obraz, který obsahuje boot sektor a je možné jej přímo zavést. V prostředí Xenu se zavádí jádro jiným způsobem a tento obraz není možné přímo využít. Aby mohlo být jádro HelenOS v prostředí Xenu zavedeno, jsou potřeba:

- Jádro samotné, které je již vytvořeno v požadovaném formátu ELF jako soubor `kernel.raw`. Jádro však musí poskytovat parametry nezbytné pro zavedení prostřednictvím ELF poznámek nebo xen guest sekce (viz 3.2.2).
- Důležité úlohy, které jsou předány jádru ve formě modulů a bez kterých není možné systém HelenOS sputit, např. ramdisk nebo úloha init.

V konfiguračním souboru domény lze uvést přímo cestu k jádru (soubor `kernel.raw`), avšak konfigurace umožňuje uvést pouze jeden modul, čili všechny nezbytné moduly bylo třeba spojit do jednoho souboru.

6.3.1 ELF poznámky

Aby bylo možné zavést jádro hypervisorem Xen, bylo nutné poskytnout parametry prostřednictvím xen guest sekce nebo ELF poznámek. ELF poznámky se jeví jako lepší řešení, jelikož není třeba převádět dynamické adresy na jejich textovou reprezentaci.

Definice ELF poznámek je umístěna v souboru `kernel/arch/ia32xen/src/boot/boot.S`. Význam jednotlivých položek je popsán v části 3.2.2.

6.3.2 Úvodní mapování

Při inicializaci jádra je třeba vytvořit úvodní mapování. Úvodní mapování identicky mapuje celou fyzickou paměť od adresy `0x80000000`. V portu na architekturu IA-32 se stará o vytvoření úvodního mapování část kódu, která je umístěna v unmapped sekci ELF souboru jádra. Zavaděč začne vykonávat v této sekci, jejíž jedinným úkolem je vytvořit úvodní mapování a skočit do mapované části.

V prostředí Xenu lze úvodní mapování nastavit parametry jádra. Toto mapování je pak vytvořeno Xenem (avšak ne celé). Úvodní mapování vytvořené Xenem je popsáno v sekci 3.2.1. Xen neprovádí namapování celé pseudo-fyzické paměti. Jedním z prvních úkolů jádra je tedy vytvořit celé identické mapování pseudo-fyzické paměti od adresy `0x80000000` rozšířením stávajícího mapování. Toto mapování je vytvořeno funkcí `pagetable_init()`, která je implementována v souboru `kernel/arch/ia32xen/src/ia32xen.c`.

Jelikož je část pseudo-fyzického paměťového prostoru již namapována, jsou tyto rámce přeskočeny a začíná se identicky mapovat až od prvního potenciálně

nenamapovaného rámce (toto je prováděno pouze kvůli optimalizaci, šlo by totiž začínat od čísla 0). Číslo tohoto pseudo-fyzického rámce je získáno tak, že se vezme číslo rámce, kde začínají zaváděcí stránkovací tabulky (`start_info.pt_base`), a k němu je přičten počet stránek, které zaváděcí stránkovací tabulky zabírají. Dále je přičteno číslo 128, což je počet stránek, které jsou garantovány Xenem, že jsou skutečně ještě namapovány. Poté je proveden pokus na namapování všech zbylých pseudo-fyzických rámců. Příslušný rámec je namapován pouze v případě, že ještě nebyl namapován Xenem.

6.3.3 Moduly

Prostřednictvím modulů se předává jádru jednak ramdisk, ale také důležité úlohy, např. init. Konfigurace Xenu však umožňuje uvést pouze jeden modul pomocí konfigurační volby ramdisk (viz [7]). Zvoleným řešením je spojení všech modulů do jednoho pomocí základních příkazů shellu. Výsledný soubor má název `modules.domU` a uvádí se v konfiguraci Xenu prostřednitvím volby ramdisk.

6.4 Konzole jádra

Konzole je v jádře implementována v souboru `kernel/arch/ia32xen/src/drivers/xconsole.c`. Implementace poskytuje jednak podporu konzole jádra, ale také pomocí databáze sysinfo zveřejňuje informace do uživatelského prostoru pro implementaci uživatelské konzole a povoluje namapování kruhového bufferu do uživatelského prostoru prostřednictvím systémového volání `physmem_map` zaregistrováním příslušné oblasti fyzické paměti.

6.4.1 Výstup

Jádro poskytuje pro implementaci základní infrastrukturu. Výstupní zařízení je v jádru systému reprezentováno strukturou `outdev_t`, se kterou jsou spojeny operace zařízení definované strukturou `outdev_operations_t`. Základní operací výstupního zařízení je operace write implementovaná funkcí `xen_console_write`, která zapíše jeden znak na výstup. Výstupní zařízení je poté přidáno k seznamu výstupních zařízení funkcí `stdout_wire`.

Funkce `xen_console_write` se nejdříve přesvědčí, že je v kruhovém bufferu místo. Pokud není v kruhovém bufferu místo, je provedeno aktivní čekání, dokud nebude místo uvolněno. Ve chvíli, kdy je v kruhovém bufferu místo, je proveden zápis, aktualizovány příslušné indexy a provedeno upozornění backend části ovladače (funkce `xen_console_notify`).

6.4.2 Vstup

Standardní vstup je inicializován funkcí `stdin_wire`. Ve chvíli, kdy přijde upozornění od backendu na nová data, jsou všechny znaky postupně zaslány na vstupní zařízení funkcí `indev_push_character` a aktualizovány indexy kruhového bufferu.

6.5 Správa paměti

Změny týkající se správy paměti byly provedeny v rámci mm subsystému jádra. Bylo implementováno rozhraní pro správu mapování, správu TLB a sdílené mezidoménové paměti.

6.5.1 Převody adres

Pro převody adresy byla definována makra, která fungují ve stejném duchu jako stávající makra PA2KA a KA2PA:

MA2PA Převod machine adresy na pseudo-fyzickou. Převod je realizován prostřednictvím převodní tabulky, která je namapována do adresového prostoru všech domén a je umístěna v horních 168MB vyhrazených pro Xen.

PA2MA Převod pseudo-fyzické adresy na machine. Převod je realizován prostřednictvím převodní tabulky pseudofyzických adres na machine, na kterou je odkazováno ze struktury `start_info`.

Makra jsou definována v souboru `kernel/arch/ia32xen/include/mm/frame.h`.

6.5.2 Mapování

Při implementaci mapování existovaly dvě možnosti, jak implementovat rozhraní systému HelenOS pro správu mapování. Toto rozhraní je definováno strukturou `page_mapping_operations_t` a obsahuje ukazatele na funkce pro vložení, nalezení a úpravu mapování.

- Použít stávající implementaci generických stránkovacích tabulek, kterou využívá port IA-32.
- Implementovat rozhraní pro správu mapování čistě pro paravirtualizovanou variantu.

Zásadní otázkou je, jestli je generická implementace stránkovacích tabulek dostatečně obecná, aby ji bylo možné použít i pro paravirtualizovanou variantu. Generická implementace na několika místech předpokládá, že může do stránkovacích tabulek přímo zapisovat. Přímý zápis do stránkovacích tabulek je však uskutečnitelný pomocí zapisovatelných stránkovacích tabulek, který lze využít společně s aktualizací tabulek přes hypervolání.

Využití generických stránkovacích tabulek má však i další problém, a to je fakt, že se jedna položka stránkovacích tabulek aktualizuje pomocí dvou zápisů. V prostředí Xenu to však znamená dvě hypervolání. Hypervolání je však relativně pomalé a technicky lze aktualizaci provést pouze v jednom hypervolání. Zejména z těchto důvodů nebyly generické stránkovací tabulky využity. Implementace byla provedena v souboru `kernel/arch/ia32xen/src/mm/page.c`.

Dále bylo třeba implementovat operace nad adresovým prostorem. Tyto operace jsou definovány strukturou `as_operations_t`, která obsahuje ukazatele na několik funkcí. První funkcí je vytvoření adresového prostoru, která vytváří nejvyšší úroveň stránkovací tabulky. Další funkcí je zrušení adresového prostoru.

Zbylé funkce souvisejí se zamykáním adresového prostoru. Implementace byla provedena v souboru `kernel/arch/ia32xen/src/mm/as.c`.

Při implementaci bylo opět možné využít buď generické implementace operací nebo implementovat operace vlastní. Každý adresový prostor musí zahrnovat identické mapování pseudo-fyzické paměti od adresy `0x80000000`. Do nově vytvářené tabulky je toto mapování generickou implementací zkopírováno z tabulek adresového prostoru jádra (zkopírována je celá horní polovina tabulky nejvyšší úrovně).

Xen však vyžaduje, aby stránka, na kterou odkazuje poslední položka stránkovací tabulky nejvyšší úrovně, nebyla sdílena více adresovými prostory. Je to z důvodu, že Xen vytváří privátní mapování pro daný adresový prostor. Přístup použitý v generické implementaci tedy nelze využít.

6.5.3 TLB

Operace nad TLB jsou implementovány v souboru `kernel/arch/ia32xen/src/mm/tlb.c`.

6.5.4 Segmentace

Xen poskytuje výchozí flat segmenty, takže operační systémy, které se segmenty přímo nepracují, nemusejí instalovat vlastní tabulky deskriptorů. V operačním systému HelenOS by tyto segmenty postačovaly také, avšak s výjimkou TLS dat.

V operačním systému HelenOS má každé vlákno (fibril) vlastní TLS segment, takže je třeba instalovat vlastní tabulku deskriptorů. Tabulka deskriptorů je definována v souboru `kernel/arch/ia32xen/src/pm.c`. Globální tabulka deskriptorů je instalována v rámci inicializace power managementu ve funci `pm_init`.

TLS segment je aktualizován při každém naplánování vlákna z uživatelského prostoru prostřednictvím systémového volání `set_tls_desc`, což umožňuje adresovat TLS data relativně vůči segmentu.

6.5.5 Sdílená paměť

Inplementaci sdílené paměti lze rozdělit na dvě části. První částí je implementace rozhraní sdílené paměti v jádře systému. Druhou částí je zpřístupnění sdílené paměti do uživatelského prostoru pro podporu ovladačů. Nejprve se podíváme na implementaci rozhraní přístupného z jádra operačního systému a poté na metodu zpřístupnění sdílené paměti do uživatelského prostoru.

Rozhraní v jádře

V jádře systému bylo implementováno základní rozhraní pro práci se sdílenou pamětí, které slouží ke správě grant tabulek. Toto rozhraní je implementováno v souboru `kernel/arch/ia32xen/src/mm/gnttab.c`. Rozhraní umožňuje povolení přístupu jiné domény k zadanému rámci buď pouze pro čtení, nebo pro čtení i zápis a opětovné zrušení přístupu:

gnttab_grant_access Povolení přístupu k danému machine rámci. Funkce vytvoří záznam v grant tabulce dle zadaných parametrů. Prvním parametrem

je id domény, pro kterou je přístup povolen. Druhý parametr je číslo sdíleného machine rámce. Dalším parametrem je určeno, jestli je rámec sdílen pouze pro čtení, nebo pro čtení i zápis. Poslední parametr vrací příslušnou grant referenci.

gnttab_end_access Ukončení sdílení paměti, jediným parametrem je grant reference. Funkce zruší platnost záznamu v grant tabulce.

Před použitím funkcí jsou grant tabulky inicializovány zavoláním gunkce `gnttab_init`. Počet rámců použitých pro grant tabulku je určen konstantou `NR_GRANT_FRAMES`. Tabulka je inicializována hypervoláním `grant_table_op` (operace `setup_table`) a poté namapována do virtuálního prostoru.

Operace nad Grant tabulkou

Nad tabulkou je třeba implementovat dvě operace. Jednak se jedná o nalezení volného záznamu pro nové sdílení (potřeba pro `gnttab_grant_access`), avšak také uvolnění konkrétního záznamu (potřeba pro `gnttab_end_access`). Uvolnění záznamu je snadné, jelikož grant reference je pouhým indexem do grant tabulky, takže stačí příslušný záznam vynulovat. Vyhledat volný záznam by však vyžadovalo projít potenciálně celou tabulku. Aby nebylo nutné procházet celou tabulku, je pomocí spojového seznamu evidován seznam volných záznamů, což umožní najití volného záznamu v konstantním čase.

Seznam volných rámců je staticky naalokován jako pole struktur `usage_info_t` o velikosti `NR_GRANT_ENTRIES`. Tento seznam rámců je indexován grant referencí, takže v případě uvolnění rámce je snadné položku opět zapojit do spojového seznamu. Pro nalezení volného záznamu stačí vzít první prvek spojového seznamu. Pokud je seznam prázdný, značí to plnou grant tabulku. Je použita generická implementace spojového seznamu systému HelenOS.

Sdílená paměť v uživatelském prostoru

Jádro samo o sobě v současné implementaci sdílenou paměť nikterak nevyužívá. Jediným využitím sdílené paměti jsou split ovladače v uživatelském prostoru. Aby měly ovladače přístup ke sdílené paměti, bylo vytvořeno nové systémové volání `xen_share_mem`.

Úloha může sdílet libovolnou část svého volného adresováho prostoru. Sdílení je provedeno prostřednictvím systémového volání `xen_share_mem`. Prvním parametrem je id domény, pro kterou je paměť sdílena. Druhým parametrem je virtuální adresa sdílené stránky. Další parametr určuje, zdali je stránka sdílena pouze pro čtení. Poslední parametr vrací grant referenci.

Systémové volání vytvoří ve virtuální paměti novou oblast o velikosti jedné stránky. Pro tuto oblast byl implementován nový backend s názvem xenshare. Backend je implementován v souboru `kernel/generic/src/mm/backend_xenshare.c`. Při vytvoření oblasti je alokován nový rámec a vytvořen příslušný záznam v grant tabulce. Samotné mapování do adresového prostoru úlohy je provedeno až v rámci ošetření výpadku stránky. Záznam z grant tabulky je odstraněn v rámci funkce destroy backendu xenshare, která je vyvolána při rušení adresového prostoru úlohy.

6.6 Výjimky

Výjimky jsou zpracovávány prostřednictvím trap tabulky a hlavní změnou týkající se výjimek je instalace trap tabulky místo nativní IDT. Handlery výjimek mohly zůstat prakticky beze změn. Implementace je provedena v souboru `kernel/arch/ia32xen/src/interrupt.c`.

Další změnou je návrat z přerušení. Původní handlery využívaly nativní instrukci `iret`, avšak v paravirtualizovaném prostředí má instrukce odlišné chování. Při zavolání instrukce `iret` jsou totiž zpracována všechna přerušení, která čekají na zpracování, a zároveň je atomicky doručování přerušení obnoveno. Procesor o událostech však nic neví, jelikož ty jsou čistě softwarovou implementací.

Postup, při kterém jsou při návratu zpracovány všechny události, poté je povoleno doručování událostí a zavolána instrukce `iret`, je sice funkční, avšak za určitých podmínek může dojít k rekurzivnímu doručení událostí a následnému přetečení zásobníku. Důvodem je okno, které vznikne mezi povolením doručování událostí a zavolání instrukce `iret`.

Tento problém je možné řešit dvěma způsoby. Prvním řešením je hypervolání `iret()`, které má požadované chování. Druhé řešení spočívá ve využití nativní instrukce `iret`, avšak případné rekurzivní volání musí být detekováno a ošetřeno.

Hypervolání `iret()` je jednoduché na použití, avšak díky režii na hypervolání pomalejší než varianta s nativní instrukcí `iret`. Pro svou jednoduchost je v implementaci použito hypervolání `iret()`.

6.7 Události

Cílem implementace událostí bylo v co největší míře využít stávající infrastrukturu pro zpracování přerušení. Z tohoto důvodu bylo v jádře implementováno rozhraní, které umožňuje napojit lokální port událostí nebo VIRQ na IRQ framework systému HelenOS.

Z uživatelského prostoru je možné alokovat nový port událostí a zároveň jej napojit na zpracování IRQ, což je typický požadavek split ovladače. Takto alokovaný port je automaticky uvolněn při ukončení úlohy. Zároveň je z uživatelského prostoru možné přes kanál událostí poslat upozornění.

Nejprve bude popsáno rozhraní pro práci s událostmi a poté blíže popsány principy implementace.

6.7.1 Rozhraní v jádře

Rozhraní pro práci s událostmi lze rozdělit na rozhraní přístupné z jádra operačního systému a rozhraní přístupné z uživatelského prostoru pomocí systémových volání, které slouží zejména pro uživatelské ovladače. V jádru bylo implementováno následující rozhraní (soubor `kernel/arch/ia32xen/src/interrupt.c`):

bind_virq_to_irq Napojí zpracování virtuálního IRQ pro daný virtuální procesor na zpracování IRQ.

bind_evtchn_to_irq Napojí zpracování kanálu událostí na zadané IRQ.

6.7.2 Rozhraní v uživatelském prostoru

Události jsou v uživatelském prostoru zpřístupněny pomocí několika systémových volání. První volání `XEN_EVENT_CHANNEL_ALLOC` slouží k alokování nového portu, na který se bude moci napojit vzdálená doména za účelem mezidoménové komunikace. Prvním parametrem je identifikátor domény, které je uděleno oprávnění se na port napojit. Druhý parametr je číslo IRQ, se kterým je kanál spojen. Aplikace si poté zaregistruje handler spojený s uvedeným číslem IRQ, přes který bude moci provést zpracování. Přes poslední parametr je vraceno číslo alokovaného portu.

Druhé systémové volání slouží k zasílání upozornění přes kanál událostí. Jedná se o `XEN_NOTIFY`, jehož jediným parametrem je číslo portu, přes který bude upozornění zasláno.

Všechny alokované kanály událostí jsou asociovány s danou úlohou a automaticky dealokovány při její ukončení.

6.7.3 Bližší pohled na implementaci v jádře

Informace o kanálech událostí jsou uloženy v poli `event_channels`, jehož prvkem je struktura `evtchn_info`, které je indexované číslem kanálu událostí. U každého kanálu událostí jsou uloženy informace, které jsou užitečné pro správné zpracování:

free Určuje, jestli je kanál událostí volný.

handler Ukazatel na handler, který je určen pro zpracování příchozí události. Je definováno několik základních typů handlerů. První z handlerů je instalován, pokud je kanál napojen na IRQ. Jedná se o `evtchn_irq_handler`. Tento handler přečte z informací o kanálu událostí číslo IRQ a vyvolá zpracování IRQ funkcí `irq_interrupt`. Další handler slouží ke zpracování IPI.

data Ukazatel na data, která jsou předána handleru. V současné implementaci není využito, avšak poskytuje možnost předat data vlastním handlerům.

irq Číslo IRQ, se kterým je kanál spojen.

vcpu Jakému virtuálnímu procesoru kanál přísluší. Tato informace je pouze pro optimalizaci, lze ji totiž zjistit pomocí hypervolání.

Zjednodušený průběh zpracování

Události jsou zpracovány přes framework přerušení systému HelenOS. Pro všechny události je vyhrazeno číslo přerušení 32, pro které je zaregistrován handler `event_handler`.

Při příchodu události vyvolá Xen callback generické zpracování přerušení zavoláním funkce `exc_dispatch` s parametrem 32. Framework přerušení vyvolá instalovaný handler přerušení `event_handler`, který zavolá funkci `do_xen_callback`.

Funkce `do_xen_callback` zjistí, které události čekají na zpracování. To se provádí ve dvou vnořených cyklech. Ve vnějším cyklu se postupně projdou všechny selectory, neboli indexy do bitového pole událostí, čekajících na zpracování. Ve vnitřním cyklu se pak prochází všechny události na indexu daném selectorem. Pro daný kanál událostí je poté vyvolán handler s ním asociovaný, který je zjištěn z informací o daném kanálu (viz pole `event_channels`).

Např. handler `evtchn_irq_handler` vyvolá zpracování IRQ. Tento handler je používán, pokud je daný kanál napojen na IRQ.

6.8 Multiprocesor

Implementace multiprocesoru sestává z inicializace a spuštění aplikačních procesorů. Systém HelenOS poskytuje za tímto účelem základní infrastrukturu. Implementace je provedena v `kernel/arch/ia32xen/src/smp/smp.c`.

6.8.1 Spuštění aplikačních procesorů

Spuštění aplikačních procesorů má na starosti speciální vlákno kmp, které je spuštěno bootstrap procesorem. Toto vlákno je umístěno v architektonicky závislé části a každá architektura jej implementuje samostatně.

Implementace vlákna kmp postupně pro všechny procesory zavolá funkci `xen_cpu_up`, která má na starosti spuštění daného procesoru. Počet procesorů byl již zjištěn v rámci volání funkce `smp_init` a uložen do proměnné config.cpu_count. Spuštění procesoru probíhá podle scénáře:

- Je provedena inicializace procesoru. Inicializace sestává z nastavení kontextu (struktura `vcpu_guest_context`). V rámci inicializace je alokována nová GDT (avšak zatím ne instalována, to je provedeno až samotným aplikačním procesorem v rámci inicializace power managementu), nastaveny výchozí stavy registrů a zkopírována trap tabulka z bootstrap procesoru. Nakonec je procesor inicializován prostřednictvím hypervolání.
- Je provedeno spuštění procesoru pomocí hypervolání, přičemž se čeká, dokud procesor není skutečně spuštěn pomocí synchronizace.
- Je inicializován časovač pro daný procesor. Tento časovač generuje periodicky přerušení každých 10ms a je určen pro plánování.

Aplikační procesor provede po svém spuštění základní inicializaci obdobně jako bootstrap procesor. V jednu chvíli je inicializován pouze jeden procesor, proto může být znovupoužit bootstrap zásobník a další datové struktury.

6.8.2 Zjištění id procesoru

Na mnoha místech je třeba zjistit id aktuálního procesoru, např. pro zakázání a povolení přerušení. Pro zjištění id aktuálně vykonávajícího virtuálního procesoru byla vytvořena funkce `smp_processor_id`.

Jádro systému HelenOS umožňuje uložit data asociovaná s procesorem, která jsou přístupná přes makro CPU. Tento mechanismus je však možné využít až ve chvíli, kdy je procesor inicializován, avšak id procsoru je třeba zjistit již při samotné inicializaci. Pro tento účel byla vytvořena proměnná `initializing_vcpu_id`, která obsahuje id procesoru, který se právě inicializuje. Pokud je CPU rovno `NULL`, zmanená to, že se procesor teprv inicializuje a funkce vrací hodnotu proměnné `initializing_vcpu_id`. Jinak vrací `CPU->arch.vcpu_id`.

Jiná možnost by byla využití TLS segmentu, avšak použitý mechanismus je velice jednoduchý a funkční.

6.8.3 IPI

HelenOS obsahuje základní infrastrukturu pro podporu IPI. Z pohledu architektury je třeba implementovat funkci `ipi_broadcast_arch` (soubor `kernel/arch/ia32xen/src/smp/ipi.c`). Parametrem funkce je číslo IPI, to je definováno architekturou prostřednictvím makra `VECTOR_TLB_SHOOTDOWN_IPI` a `VECTOR_DEBUG_IPI`. V současné implementaci je podporováno pouze první z uvedených.

V případě zavolání funkce `ipi_broadcast_arch` s parametrem `VECTOR_TLB_SHOOTDOWN_IPI` je generickou částí očekáváno, že bude na všech procesorech vyvolána funkce `tlb_shootdown_ipi_recv`, která se postará o zpracování.

IPI je podporováno Xenem prostřednictvím kanálů událostí. Handler `tlb_shootdown_ipi` je instalován přímo jako handler kanálu událostí, který vyvolá funkci `tlb_shootdown_ipi_recv`, tím je zaručeno odpovídající zpracování při příchodu události.

Pro zaslání IPI na všechny procesory (broadcast) je nutné převést číslo IPI na číslo portu kanálu událostí, přes který ja zasláno upozornění. Tento převod je jiný pro každý procesor, jelikož pro každý procesor je alokován jiný port. Převodní tabulka je uložena ve struktuře procesoru `cpu_arch_t`, položka `ipi_to_evtchn`. Funkce `ipi_broadcast_arch` projde všechny procesory a pro každý provede převod IPI na kanál událostí, na který zašle upozornění, to je poté zpracováno způsobem uvedeným v předchozím odstavci.

6.9 Správa procesů

Ve správě procesů bylo nutné implementovat několik změn. První změna se týká přepínání kontextu, jelikož port IA-32 využíval TSS, který není pod Xenem nikterak virtualizován. Další změna byla zapříčiněna způsobem přístupu k TLS datům v uživatelském prostoru. Nutné bylo rovněž upravit systémová volání.

6.9.1 Přepínání kontextu

Každé vlákno v systému HelenOS má jak uživatelský, tak jaderný zásobník. Pokud přijde přerušení, přepne se vlákno do úrovně oprávnění jádra a začne vykonávat obsluhu přerušení. V tento okamžik se přepíná vlákno na jaderný zásobník, toto přepnutí je provedeno atomicky procesorem. V nativním prostředí je adresa a segment zásobníku přečten z TSS segmentu. Ve virtualizovaném prostředí je tento mechanismus nahrazen hypervoláním `stack_switch`, který je volán vždy před spuštěním vlákna a nahrazuje tak funkci TSS segmentu. Toto je provedeno v rámci funkce `before_thread_runs_arch`, viz soubor `kernel/arch/ia32xen/src/proc/scheduler.c`.

6.9.2 Thread Local Storage

Xen využívá k ochraně paměti použité hypervizorem segmentaci. Důsledkem je, že guest nemá segmenty o velikosti 4GB, což přináší komplikace pro TLS. Mechanismus TLS použitý v gcc používá pro adresování TLS dat segmenty, které přetékají adresový prostor, tzv. wrap-around segmenty. Tento mechanismus je obecně

korektní a lze například vytvořit segment velikosti 4GB, který začíná v půlce adresového prostoru, přičemž jej přetéká a pokračuje od adresy `0x00000000`.

Jedno z možných řešení poskytuje sám Xen a tím je emulace 4GB segmentů. Toto řešení však vede k extrémnímu zpomalení přístupu k TLS datům.

Další možností je použití volby kompilátoru gcc `-mno-tls-direct-seg-refs` (viz [10]). Tato volba zajistí, že se k TLS datům nebude přistupovat přes wrap-around segmenty. Důsledkem toho však je, že pro paravirtualizovanou verzi budeme mít jinou libc.

6.9.3 Systémová volání

Systém HelenOS využívá dva možné mechanismy přerušení. Prvním způsobem je instrukce `SYSENTER`, druhým přerušení `0x30`. Instrukci `SYSENTER` Xen žádným způsobem neemuluje a není ji tedy možné použít. Pod Xenem není rovněž možné využít jiné softwarové přerušení než `0x80`, které je určeno pro systémová volání. Pro systémové volání je tedy využíváno softwarové přerušení `0x80`, které je zpracováno prostřednictvím trap tabulky.

6.10 Xenstore

Xenstore je implementován čistě v uživatelském prosotru jako nová úloha s názvem Xenbus (zdrojové soubory jsou v adresáři `uspace/srv/xenbus/`). S Xenbusem se komunikuje prostřednictvím IPC, avšak aplikace nemusí volat IPC metody přímo, ty využívají pro přístup knihovnu Xenbus (zdrojové soubory jsou v adresáři `uspace/lib/xenbus/`).

6.10.1 Úloha Xenbus

Úloha Xenbus je spouštěna automaticky úlohou init a zajišťuje rozhraní k databázi Xenstore. Podporovanými operacemi jsou read, write a ls, které jsou dostupné přes IPC.

Základní informace, které jsou uvedeny ve start info stránce, jsou zveřejněny do uživatelského prostoru prostřednictvím sysinfo minimalistickým ovladačem umístěným v jádře systému (soubor `kernel/arch/ia32xen/src/drivers/xbus.c`). Konkrétně se jedná o klíče:

xen.store.pa Pseudo-fyzická adresa obsahující rozhraní pro Xenstore.

xen.store.evtchn Kanál událostí použitý pro notifikaci o událostech.

xen.store.irq Číslo IRQ, které je spojeno s kanálem událostí.

Úloha Xenbus v rámci své inicializace (funkce `xenbus_init`) provede mimo jiné:

- Namapuje do svého adresového prostoru stránku s rozhraním do Xenstore, adresa je zveřejněna klíčem xen.store.pa v sysinfo. Stránka je namapována pomocí stávajícího rozhraní pro ovladače funkcí `physmem_map`.
- Ze sysinfo zjistí číslo IRQ, které je spojeno s doručováním událostí spojených s Xenstore, a zaregistruje handler `xbus_event_handler`, který bude zpracovávat příchozí události.

Nízkoúrovňové rozhraní Xenstore je asynchnonní. Nejdříve je třeba zapsat do kruhového bufferu požadavek a odeslat upozornění. Ve chvíli, kdy bude požadavek zpracován a zapsána odpověď, přijde notifikace ve formě přerušení. Úloha udržuje seznam odeslaných požadavků v poli `request_info`, které je indexováno pomocí id požadavku. U každého požadavku je uloženo:

- Informace, jestli je požadavek s daným id odeslán.
- Podmíněná proměnná, na které se čeká na odpověď.
- Data odpovědi.

Ve chvíli, kdy přijde přerušení, se provedou následující kroky:

1. Zjistí se, jestli je v kruhovém bufferu celá odpověď, pokud ne, dál se nic neprování a provede se návrat z přerušení (čeká se na další notifikaci).
2. Podle hlavičky se přečte velikost odpovědi a alokuje se paměť pomocí malloc. Do alokované paměti se poté zkopírují data odpovědi. Ukazatel je uložen do informací o požadavku.
3. Je probuzeno vlákno čekající na odpověď.
4. Pokračuje se bodem jedna.

Obecný typ požadavku lze zaslat funkcí `xbus_write_request_and_wait`, která odešle požadavek a počká na odpověď. Tato funkce je využita pro implementaci dalších požadavků. Funkce pro zaslání obecného typu požadavku má tři parametry. Prvním parametrem je typ požadavku (výčtový typ `xsd_sockmsg_type` je definován v hlavičkových souborech Xenu). Druhým parametrem je id požadavku. Id požadavku je třeba vyhradit pomocí funkce `xbus_allocate_id`, po vrácení odpovědi zase uvolnit pomocí funkce `xbus_release_id`. Dalším parametrem jsou data vlastního požadavku předaná pomocí struktury `write_request_data_t`. Data jsou pouze řetězcem a šlo by je předávat čistě jako řetězec, avšak místo pro tento řetězec by se muselo alokovat a poté spojit jednotlivé části tak, aby vznikl řetězec reprezentující výsledný požadavek. Zvolené řešení umožňuje použít několik samostatných řetězců, které jsou pak za sebou nakopírovány do kruhového bufferu.

6.10.2 Knihovna Xenbus

Aby aplikace využívající Xenstore nemusely volat IPC metody přímo, vznikla knihovna Xenbus, která zaobaluje volání IPC metod a zároveň poskytuje další užitečné funkce. Kód knihovny se nachází v adresáři `uspace/lib/xenbus/`. Knihovna poskytuje několik metod pro práci s databází Xenstore.

xbus_read Přečte a vrátí kodnotu daného klíče.

xbus_read_int Přečte a vrátí hodnotu daného klíče jako číslo. V databázi musí být číslo uloženo ve formě řetězce a musí být kladné. Pokud číslo nelze naparsovat, vrátí zápornou hodnotu.

xbus_printf Zápis na daný klíč ve formě printf. Pouze pro zjednodušení použití, jedná se o zaobalení funkce `xbus_write`.

xbus_write Zapíše hodnotu daného klíče.

xbus_wait_for_change_int Počká, dokud není hodnota klíče odlišná od zadaného. V současné implementaci je použito aktivní čekání, avšak dobudoucna se počítá s rozšířením, které bude využívat notifikace zasílané Xenstorem při změně klíče.

xbus_ls Vrátí seznam podklíčů daného klíče.

xbus_switch_state Přepne stav zařízení uložený na daném klíči a vrátí aktuální.

6.11 Uživatelská konzole

Konzoli v uživatelském prostoru lze rodělit na dvě prakticky nezávislé části – vstup a výstup. Vstup je realizován prostřednictvím input serveru, kdežto výstup pomocí framebuffer serveru. Informace jsou do uživatelského prostoru předány ovladačem konzole jádra systému prostřednictvím databáze sysinfo.

6.11.1 Informace předané jádrem

Do uživatelského prostoru jsou předány informace o konzoli prostřednictvím sysinfo. Prostřednictvím klíče `xen.console.pa` je předána pseudo-fyzická adresa sdílené paměti pro komunikaci s backend částí konzole. Pro tuto oblast je povoleno namapování do uživateslkého prostoru. Klíč `xen.console.evtchn` a `xen.console.irq` určují kanál událostí a IRQ, na které je zpracování událostí napojeno.

6.11.2 Vstup

Vstup je realizován prostřednictvím input serveru. Port pro Xen do input serveru je implementován v souboru `uspace/srv/hid/input/port/xen.c`. V rámci inicializace portu je namapován kruhový buffer konzole do prostoru úlohy a zaregistrován IRQ handler. V rámci obsluhy handleru přerušení jsou přečtena data z kruhového bufferu a předána input serveru k dalšímu zpracování prostřednictvím funkce `kbd_push_data`.

6.11.3 Výstup

Výstup je realizován prostřednictvím framebuffer serveru. Server umožňuje přidávat nové porty. Port pro Xen je implementován v souboru `uspace/srv/hid/-fb/port/xen.c`.

Pro výstup bylo možné použít stávající podporu systému pro sériovou linku podporující protokol VT100. Inicializace Xen framebuffer portu je provedena funkcí `xen_init`, která provede namapování kruhového bufferu do adresového prostoru úlohy a poté inicializuje sériový výstup pomocí funkce `serial_init`.

Funkce `serial_init` vyžaduje jako své parametry ukazatele na funkce, které zapíší příslušná data na výstupní zařízení, v případě Xenu se jedná o zápis do kruhového bufferu. Tyto funkce jsou implementovány obdobným způsobem jako v případě konzole pro jádro. Nejprve se počká, až je v kruhovém bufferu místo, poté je proveden zápis a upozornění backend části ovladače na nová data.

6.12 Bloková zařízení

Přístup k virtuálním blokovým zařízením Xenu je zprostředkován ovladačem `xen_bd`, zdrojové kódy jsou umístěny v adresáři `uspace/srv/bd/xen_bd/`. Ovladač je implementován jako samostatná úloha.

6.12.1 Inicializace

V rámci inicializace ovladače jsou nejprve rozpoznána všechna bloková zařízení. Informace o zařízení jsou přečtena z databáze Xenstore. Každé zařízení je poté inicializováno (následující akce jsou provedeny funkcí `init_device`, ne nutně v tomto pořadí):

- Je vytvořena struktura typu `blkfront_dev_t` udržující informace o zařízení, je inkrementována globální proměnná `disks_count` a ukazatel na strukturu zařízení přidán do globálního pole `disks`.
- Je vytvořena sdílená paměť a inicializován kruhový buffer pro komunikaci s backend částí ovladače.
- Je zaregistrován IRQ handler pro příchozí přerušení, alokován nový kanál událostí a spojen s číslem přerušení.
- Do Xenstoru jsou zapsány údaje, které backend potřebuje pro komunikaci s frontend částí, jedná se o číslo kanálu událostí, grant referenci sdílené paměti pro komunikaci a protokol, kterým se bude komunikovat.
- Stav zařízení je nastaven na connected a čeká se na změnu stavu backend části ovladače, dokud není také connected.
- Jsou přečteny základní údaje o geometrii disku a uloženy do informací o zařízení. Jedná se o počet bloků a velikost bloku.
- Zařízení je zaregistrováno u loc servisu.

6.12.2 Implementace IPC metod blokového zařízení

Ovladač poskytuje prostřednictvím IPC několik metod. Jedná se o zjištění počtu bloků, zjištění velikosti bloku, přečtení bloku a zápis bloku.

V rámci IPC je předána identifikace blokového zařízení, nad kterým je operace provedna. Operace zjištění počtu bloků a velikosti bloku nevyžaduje žádnou interakci s backend částí ovladače. Tyto informace jsou totiž zjištěny a uloženy při inicializaci ovladače z databáze Xenstore.

Pro implementaci operace přečtení a zápis bloku byla vytvořena funkce `xen_bd_rw_blocks`. Prvním parametrem je struktura `blkfront_dev_t` reprezentující zařízení, nad kterým je operace provedena. Druhý parametr určuje, jestli jde o čtení nebo zápis. Třetím parametrem je číslo sektoru. Čtvrtý parametr určuje počet sektorů a poslední parametr je adresa bufferu, do kterého budou data buď zapsána, nebo z něj přečtena (dle operace).

Komunikace s backend částí

Komunikace probíhá standardně přes sdílenou paměť, upozornění jsou zasílána pomocí kanálu událostí. Při zaslání požadavku je alokována struktura ty-

pu `request_info_t`, která obsahuje základní informace o požadavku. Jedná se zejména o synchronizační primitivum `fibril_condvar_t`, na kterém se čeká na asynchronní odpověď. Dále je alokován prostor v kruhovém bufferu a zapsán požadavek. Jako id požadavku je uvedena adresa alokované struktury `request_info_t`. Po zaslání požadavku je provedeno čekání na odpověď.

Při příchodu přerušení (handler přerušení `xen_irq_handler`) je přečtena odpověď, zjištěno id požadavku. Toto id je ve skutečnosti adresa struktury, která obsahuje informace o požadavku. Do informací o požadavku je zapsán výsledek a probuzen fibril čekající na odpověď.

6.13 Debugger

V kontextu Xenu lze uvažovat o debugování různých komponent systému, a to různými metodami, přesto bude zmíněn pouze princip debugování DomU jádra prostřednictvím GDB a to z důvodu, že byla tato metoda zejména v prvních fázích portu velice užitečná.

Aby bylo možné využít všech funkcí debuggeru, musí být jádro systému HelenOS sestaveno s podporou debuggování. Poté je vytvořena příslušná doména a spuštěn GDB server (na doméně nula):

```
# xm create /cesta/k/helenos.cfg --pause
# gdbsx -a 1 32 9999
```

Poté je spuštěno GDB, načteny debuggovací symboly a je provedeno spojení s GDB serverem:

```
# gdb
# (gdb) file /cesta/k/helenos/kernel/kernel.raw
# Reading symbols from /cesta/k/helenos/kernel/kernel.raw...done.
# (gdb) target remote <vzdálená ip>:9999
# Remote debugging using 127.0.0.1:9999
```

7. Závěr

Práce měla dva na sobě prakticky nezávislé cíle. Prvním cílem byl popis rozhraní hypervisoru Xen z pohledu paravirtualizovaného operačního systému, který pod hypervisorem běží. Tento cíl je splněn v rámci kapitoly 3. Jelikož je rozhraní hypervisoru relativně rozsáhlé, je popis zaměřen zejména na potřeby neprivilegované domény.

Druhým cílem byla implementace a popis úprav operačního systému HelenOS pro běh v paravirtualizovaném prostředí Xenu. Výstupem práce je funkční port operačního systému HelenOS na platformu Xen. Prototypová implementace je provedena pro architekturu IA-32. Práce obsahuje popis relevantních částí systému HelenOS (kapitola 4), analýzu a popis provedených změn (kapitoly 5 a 6).

Byly implementovány všechny vlastnosti, které jsou nezbytné pro paravirtualizovaný běh operačního systému, včetně podpory pro SMP a blokových zařízení. V uživatelském prostoru byla vytvořena kompletní infrastruktura pro implementaci dalších frontend ovladačů, které však nebyly zejména z časových důvodů implementovány.

Průběh práce velice zpomalovala neúplná a zastaralá dokumentace hypervisoru Xen. Při vypracování bylo mnohdy nezbyté postupovat netodou pokus-omyl. Nepostradatelným zdrojem informací byly rovněž zdrojové kódy hypervisoru a portu operačního systému Linux a MiniOS.

7.0.1 Možnosti rozšíření

Z možností dalšího rozšíření ze nabízí např. podpora dalších architektur, zejména AMD64 a IA-64, které jsou podporovány hypervisorem Xen a v nativní verzi rovněž systémem HelenOS. Dalším možným rozšířením je implementace různých dosud neimplementovaných frontend a backend ovladačů, popř. domény nula.

Literatura

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

[2] 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://www.usenix.org/events/sec2000/robin.html`

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

[4] Lenka Trochtová, *Rozhraní pro ovladače zařízení v HelenOS*, diplomová práce, Univerzita Karlova v Praze, 2010

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

[6] The Xen Team, *Xen Interface Manual, Xen v3.0 for x86*, University of Cambridge, 2005

[7] The Xen Team, *Users' Manual, Xen v3.3*, University of Cambridge, 2008

[8] *IPC for Dummies*, dostupné on-line `http://trac.helenos.org/wiki/IPC`

[9] *Compiling HelenOS From Source*, dostupné on-line `http://trac.helenos.org/wiki/UsersGuide/`

[10] *XenSpecificGlibc*, dostupné on-line `http://wiki.xensource.com/xenwiki/XenSpecificGlibcCompilingFromSource`

[11] *Vendor-specific ELF Note Elements*, dostupné online `http://www.netbsd.org/docs/kernel/elf-notes.html`

[12] *Architecture for Split Drivers Within Xen*, dostupné online `http://wiki.xen.org/xenwiki/XenSplitDrivers`

A. Obsah přiloženého CD

K práci je přiloženo CD, které obsahuje:

images/ Adresář obsahující spustitelný obraz s předinstalovaným hypervisorem Xen a doménou 0 pro Qemu, resp. KVM. Použití obrazu je popsáno v části B.1.

src/ Přiložené zdrojové kódy.

- **helenos-xen.tar.gz** Zabalené zdrojové kódy systému HelenOS, potažmo této práce.
- **xen-4.1.1.tar.gz** Zabalené zdrojové kódy hypervisoru Xen 4.1.1.

doc/ Adresář obsahující text této práce v elektronické podobě.

- **tex/** Text této práce ve formátu TeX.
- **pdf/** Text této práce ve formátu PDF.

B. Spuštění HelenOS jako DomU

V této části jsou uvedeny dva postupy, jak spustit systém HelenOS jako DomU. První postup spočívá ve využití obrazu, který je součástí přiloženého CD. Tento obraz obsahuje vše potřebné. Druhý postup popisuje spuštění přímo pomocí zdrojových kódů, avšak předpokládá již nainstalovanou doménu 0.

B.1 Z přiloženého obrazu

Dejme tomu, že máme cdrom připojen do adresáře `/cdrom`. Nejprve rozbalíme obraz někam na disk, např. do `/tmp/debian-xen.img`.

```
gunzip -c /cdrom/images/debian-xen.img.gz > /tmp/debian-xen.img
```

Poté vytvoříme virtuální stroj pomocí Qemu.

```
qemu -hda /tmp/debian-xen.img -m 512
```

Pro urychlení je možné použít rovněž KVM, které využívá úplnou virtualizaci místo simulace.

```
kvm -hda /tmp/debian-xen.img -m 512
```

Vyčkáme, až ve virtuálním stroji nabootuje systém Linux (v bootloaderu Grub je přednastavena správná volba). Poté se přihlásíme s následujícími údaji:

```
login: root
heslo: helenos
```

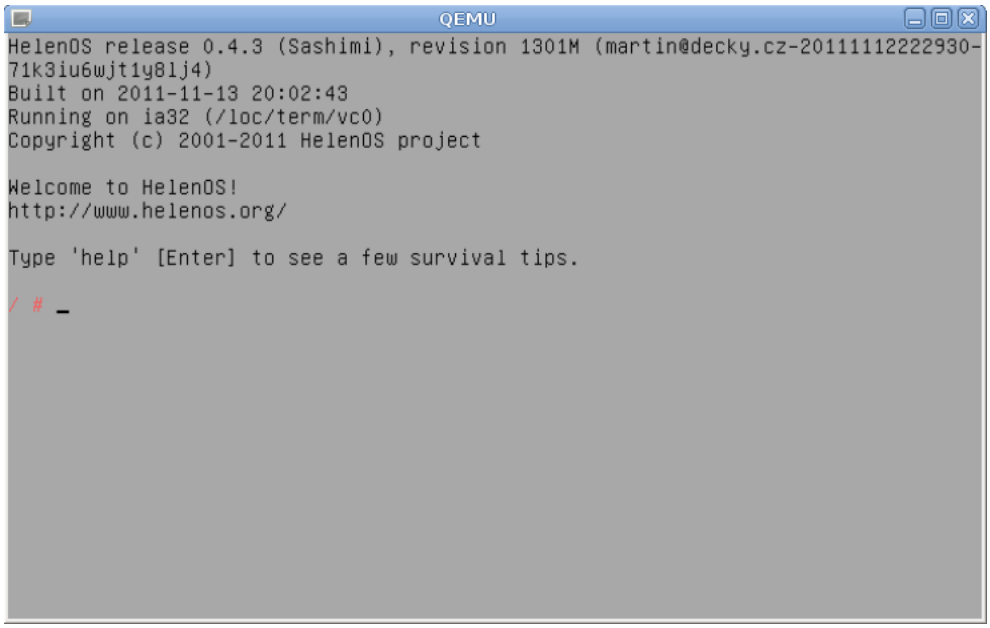
Po přihlášení je aktuálním adresářem `/root`, ve kterém se nachází všechny potřebné soubory. Spuštění HelenOS jako DomU se provede příkazem:

```
xm create -c helenos.cfg
```

Doména má nakonfigurovaný jeden pevný disk a 2 virtuální procesory. Spuštěnou doménu zobrazuje obrázek B.1. Doménu lze vypnout z jiného terminálu příkazem `xm destroy helenos`.

B.2 Ze zdrojových kódů

Návod předpokládá již nainstalovaný hypervisor Xen a doménu 0. Předpokládá se, že všechny uvedené příkazy budou spouštěny z domény 0 a doménou 0 bude Linux. Pro spuštění HelenOS jako DomU je neprve třeba systém překompilovat a poté vytvořit konfigurační soubor domény pro Xen.



Obrázek B.1: HelenOS jako DomU

B.2.1 Kompilace

Zdrojové soubory systému HelenOS se nachází na přiloženém CD. Kompilace se spouští z kořenové složky zdrojových souborů příkazem `make`. Pokud je kompilace spouštěna poprvé, je zobrazena konfigurace systému (tu lze také vyvolat příkazem `make config`). Pro konfiguraci je možné využít přednastavené hodnoty. Pro načtení těchto voleb zvolte v podmenu `Load preconfigured defaults` volbu `ia32xen`.

Výstupem jsou dva soubory, které jsou důležité pro konfiguraci domény. Jedná se o `kernel/kernel.raw` a `distroot/boot/modules.xenU`.

B.2.2 Konfigurační soubor domény

Následující ukázka konfiguračního souboru hypervisoru Xen pro HelenOS DomU ukazuje pouze základní použití. Detailní popis konfiguračních voleb lze nalézt v uživatelském manuálu Xenu [7]. Při použití je třeba upravit cesty dle aktuálního umístění. V uvedeném konfiguračním souboru je nastavena cesta ke zkompilovaném jádru domény a ramdisku. Dále je nastavena velikost paměti, název domény, počet virtuálních procesorů a jeden virtuální disk, který je ve skutečnosti souborem v doméně 0.

```
kernel = "/home/tomas/dipl/HelenOS/kernel/kernel.raw"
ramdisk = "/home/tomas/dipl/HelenOS/boot/distroot/boot/modules.xenU"

memory = 258
name = "helenos"
vcpus = 2
disk = [ 'file:/home/tomas/dipl/disk.img,xvda,w' ]

on_crash = "destroy"
```

B.2.3 Virtuální disk

Virtuální disky jsou nastaveny v konfiguračním souboru pomocí konfigurační direktivy `disk`. V ukázce je disk, jehož backendem je soubor. Tento soubor je možné vytvořit pomocí příkazů:

```
# dd if=/dev/null of=disk.img bs=1M seek=1024
# mkfs.vfat disk.img
```

Velikost disku je dána parametrem `seek` příkazu `dd`. Příkazem `mkfs.vfat`[1] je vytvořen souborový systém. V konfiguraci systému HelenOS je třeba nastavit podporu souborového systému FAT a blokových zažízení.

[1] V operačním systému Debian je příkaz součástí balíčku `dosfstools`

C. Bootovací zprávy Xenu

Následující výpis zobrazuje bootovací zprávy Xenu. Nejprve je provedena inicializace Xenu, poté vytvoření domény nula a nakonec její spuštění.

```
(XEN) Xen version 4.1.1 (Debian 4.1.1-2) (waldi@debian.org) (gcc version 4.6.1 ↩
    (Debian 4.6.1-5) ) Sat Aug  6 10:19:30 UTC 2011
(XEN) Bootloader: GRUB 1.99-12
(XEN) Command line: placeholder
(XEN) Video information:
(XEN)  VGA is text mode 80x25, font 8x16
(XEN)  VBE/DDC methods: V2; EDID transfer time: 1 seconds
(XEN) Disc information:
(XEN)  Found 2 MBR signatures
(XEN)  Found 2 EDD information structures
(XEN) Xen-e820 RAM map:
(XEN)  0000000000000000 - 000000000009fc00 (usable)
(XEN)  000000000009fc00 - 00000000000a0000 (reserved)
(XEN)  00000000000e4000 - 0000000000100000 (reserved)
(XEN)  0000000000100000 - 00000000bff90000 (usable)
(XEN)  00000000bff90000 - 00000000bff9e000 (ACPI data)
(XEN)  00000000bff9e000 - 00000000bffd0000 (ACPI NVS)
(XEN)  00000000bffd0000 - 00000000bffde000 (reserved)
(XEN)  00000000bffe0000 - 00000000c0000000 (reserved)
(XEN)  00000000fee00000 - 00000000fee01000 (reserved)
(XEN)  00000000fff00000 - 0000000100000000 (reserved)
(XEN) System RAM: 3071MB (3144892kB)
(XEN) ACPI: RSDP 000FB630, 0014 (r0 ACPIAM)
(XEN) ACPI: RSDT BFF90000, 003C (r1 A_M_I_ OEMRSDT   8001020 MSFT       97)
(XEN) ACPI: FACP BFF90200, 0084 (r2 A_M_I_ OEMFACP   8001020 MSFT       97)
(XEN) ACPI: DSDT BFF905C0, 7C37 (r1  A0846 A0846000        0 INTL 20060113)
(XEN) ACPI: FACS BFF9E000, 0040
(XEN) ACPI: APIC BFF90390, 006C (r1 A_M_I_ OEMAPIC   8001020 MSFT       97)
(XEN) ACPI: MCFG BFF90400, 003C (r1 A_M_I_ OEMMCFG   8001020 MSFT       97)
(XEN) ACPI: OEMB BFF9E040, 0080 (r1 A_M_I_ AMI_OEM   8001020 MSFT       97)
(XEN) ACPI: HPET BFF98200, 0038 (r1 A_M_I_ OEMHPET   8001020 MSFT       97)
(XEN) ACPI: GSCI BFF9E0C0, 2024 (r1 A_M_I_ GMCHSCI   8001020 MSFT       97)
(XEN) Xen heap: 9MB (9788kB)
(XEN) Domain heap initialised
(XEN) Processor #0 7:7 APIC version 20
(XEN) Processor #1 7:7 APIC version 20
(XEN) Processor #2 7:7 APIC version 20
(XEN) Processor #3 7:7 APIC version 20
(XEN) IOAPIC[0]: apic_id 4, version 32, address 0xfec00000, GSI 0-23
(XEN) Enabling APIC mode:  Flat.  Using 1 I/O APICs
(XEN) Table is not found!
(XEN) Using scheduler: SMP Credit Scheduler (credit)
(XEN) Detected 2745.588 MHz processor.
(XEN) I/O virtualisation disabled
(XEN) ENABLING IO-APIC IRQs
(XEN)  -> Using new ACK method
(XEN) Platform timer is 14.318MHz HPET
(XEN) Allocated console ring of 16 KiB.
(XEN) VMX: Supported advanced features:
(XEN)  - APIC MMIO access virtualisation
(XEN)  - APIC TPR shadow
(XEN)  - Virtual NMI
(XEN)  - MSR direct-access bitmap
(XEN) HVM: ASIDs disabled.
(XEN) HVM: VMX enabled
(XEN) Brought up 4 CPUs
(XEN) *** LOADING DOMAIN 0 ***
(XEN)  Xen  kernel: 32-bit, PAE, lsb
(XEN)  Dom0 kernel: 32-bit, PAE, lsb, paddr 0x1000000 -> 0x1790000
(XEN) PHYSICAL MEMORY ARRANGEMENT:
(XEN)  Dom0 alloc.:   00000000ba000000->00000000bc000000 (730501 pages to be ↩
    allocated)
(XEN)  Init. ramdisk: 00000000be5fb000->00000000bfdff600
(XEN) VIRTUAL MEMORY ARRANGEMENT:
(XEN)  Loaded kernel: c1000000->c1790000
```

```
(XEN)  Init. ramdisk: c1790000->c2f94600
(XEN)  Phys-Mach map: c2f95000->c326c628
(XEN)  Start info:    c326d000->c326d47c
(XEN)  Page tables:   c326e000->c328d000
(XEN)  Boot stack:    c328d000->c328e000
(XEN)  TOTAL:         c0000000->c3400000
(XEN)  ENTRY ADDRESS: c1424000
(XEN) Dom0 has maximum 4 VCPUs
(XEN) Scrubbing Free RAM: .done.
(XEN) Xen trace buffers: disabled
(XEN) Std. Loglevel: Errors and warnings
(XEN) Guest Loglevel: Nothing (Rate-limited: Errors and warnings)
(XEN) Xen is relinquishing VGA console.
(XEN) *** Serial input -> DOM0 (type 'CTRL-a' three times to switch input to ↩
    Xen)
(XEN) Freed 180kB init memory.
```