**ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE**

**FAKULTA PROVOZNĚ EKONOMICKÁ**

**KATEDRA INFORMAČNÍHO INŽENÝRSTVÍ**


# Diplomová práce

## *Virtualizace*

**Vypracoval:** Filip Horák

**Vedoucí diplomové práce:** Ing. Martin Papík, Ph.D.

# P r o h l á š e n í

Prohlašuji, že jsem diplomovou práci na téma Virtualizace vypracoval samostatně a použil k tomu pouze literaturu, kterou uvádím v seznamu přiloženém k diplomové práci.

Nemám námitky proti půjčování, zveřejnění a dalšímu využití práce, pokud s tím bude souhlasit Katedra informačního inženýrství Provozně ekonomické fakulty České zemědělské univerzity v Praze.

Filip Horák

……

V Praze dne …….

# Poděkování

Děkuji především vedoucímu své diplomové práce panu Ing. Martinovi Papíkovi, Ph.D. za podnětné konzultace, poskytnuté materiály a pozornost, kterou věnoval mé práci. Dále děkuji všem, kteří mi při psaní mé diplomové práce nějakým způsobem pomohli, například zapůjčením materiálů či cennými radami a připomínkami. Velmi také děkuji Michalu Slancovi, který mi umožnil provedení praktické části ve společnosti Atelier Slanec, s.r.o.

# *Virtualizace*

## *Virtualization*

## Souhrn

Tato práce se zabývá problematikou virtualizace počítačové infrastruktury. Jsou v ní definovány základní termíny týkající se virtualizace, zahrnující funkci hypervizoru, činnost procesoru, paměti a vstupně-výstupních zařízení. Dále jsou v práci popsány a zhodnoceny nejběžnější virtualizační techniky a srovnány jejich parametry. V další části je pak zhodnocen trh virtualizačních řešení a jeho hlavní hráči. Praktickým výstupem této práce je popis implementace virtualizace serveru ve společnosti Atelier Slanec, s.r.o.

## Summary

This work deals with the issue of virtualization of computer infrastructure. There are defined basic terms related to virtualization, including functionality of hypervisor, activity of processor, memory unit and input-output devices. Further, there are described and evaluated the most common virtualization techniques, and compared their characteristics. In the next section is evaluated market of virtualization solutions and its major players. The practical outcome of this work is the description of the implementation of server virtualization at Atelier Slanec, Ltd.

## Klíčová slova

Virtualizace, hypervizor, paravirtualizace, virtuální stroj, emulace, procesor, paměť, I/O zařízení, host, hostitel

## Key words

Virtualization, hypervisor, paravirtualization, virtual machine, emulation, processor, memory, I/O device, guest, host

# Obsah

# 1 Úvod

Virtualizace je pojem, který se v posledních letech často skloňuje. Důvodem je především skutečnost, že nasazení vhodného virtualizačního řešení může přinést kvalitnější výsledek, při značném snížení nákladů. Jelikož současná společnost využívá informační technologie čím dál intenzivněji, nelze si již v některých oblastech lidské činnosti představit efektivní fungování bez nasazení virtualizovaných řešení.

Metody virtualizace serverů jsou sice využívány již několik desítek let, donedávna však byly doménou především velkých organizací. S příchodem virtualizačních produktů pro počítače architektury x86 však virtualizace začala pronikat i mezi střední a malé podniky. Díky stále klesajícím cenám hardwarového vybavení a jednoduchým a stabilním virtualizačním řešením se tak virtualizace objevuje i tam, kde by to dříve nebylo možné.

Pojem virtualizace je velmi obecný, a tak není překvapením, že v sobě skrývá množství různých přístupů a technik. Není v možnostech jedné práce popsat a seznámit čtenáře se všemi významy tohoto slova a různých trendů, přístupů a možností, které se za ním skrývají. Tato práce se tak zaměřuje na poměrně úzkou skupinu virtualizačních řešení, která poskytují možnosti virtualizace jednotlivých serverů a v menší míře i desktopových klientů. Záměrně jsou tak opomíjeny v poslední době často zmiňovaná cloudová řešení a grid computing.

Práce tak definuje pouze obecné metody virtualizace umožňující přistupovat k jednomu fyzickému zařízení několika logickým entitám. Po nezbytném definování základních podmínek virtualizace a jejích parametrů, je tak čtenáři připraven souhrn nejčastěji používaných virtualizačních technik využívaných k tomuto přístupu. Pro úplnost pak nelze opomenout základní analýzu trhu s virtualizačními produkty a stručné srovnání největších výrobců a jejich produktů.

# 2 Cíl a metodika

## 2.1 Cíl práce

Cílem této práce je popsat možnosti virtualizace operačních systémů na architektuře x86. Práce má definovat základní teoretická východiska, týkající se práce hypervizoru a virtualizace procesoru, paměti a vstupně-výstupních zařízení. Následným cílem je provést komparaci nejpoužívanějších virtualizačních technik, se zvýrazněním jejich předností a omezení. Poté by autor rád provedl stručný popis trhu virtualizačních řešení a nejvýznamnějších společností na něm působících. Práce by také měla přiblížit hlavní vlastnosti nejpoužívanějších virtualizačních produktů.

V druhé části práce si autor bere za cíl popsat konkrétní implementaci virtualizace serverové technologie v malém podniku. Uvézt přednosti tohoto řešení a úskalí, s kterými se při realizaci setkal.

## 2.2 Metodika

Informace pro tuto práci byly použity z volně dostupných zdrojů. Vzhledem k odbornosti tématu a nedostatku literatury v češtině, byla výrazná většina zdrojů cizojazyčných. Některé informace byly získány z internetových zdrojů, jelikož se nevyskytují v tištěné podobě. To se týká především informací o virtualizačních produktech.

V rámci metodického aparátu byla v první půlce práce využita především literární rešerše. Ve druhé části pak bylo použito několik srovnávacích analýz, z nichž nejvýznamnější byla analýza Magic Quadrants společnosti Gartner.

Zásadní přínos k práci měla také činnost realizovaná ve společnosti Atelier Slanec, s.r.o. V rámci této činnosti byly použity standardní postupy a techniky, které jsou v souvislosti se zaváděním virtualizace používány.

# 3 Teoretická východiska

## 3.1 Definice virtualizace

Jedná se o množinu postupů a technik, které díky určité míře abstrakce umožňují provozování několika logických systémů v rámci jednoho fyzického systému a/nebo naopak několik fyzických systémů jako jeden logický celek. K tomu využívají metody, které umožňují efektivní dělení (nebo agregaci) zdrojů. Virtualizovat se dají jak celé platformy, tak jejich jednotlivé hardwarové nebo softwarové komponenty. Nezřídka se pak obě virtualizační techniky propojují a vzniká výkonné serverové pole, v jehož rámci se poté virtualizují jednotlivé logické subsystémy.

Tato práce se však zabývá pouze virtualizací v rámci jednoho fyzického systému a jím odvozených technik. Dále se také, až na pár výjimek, zabývá pouze virtualizací na platformě x86 respektive x86_64.

## 3.2 Přínosy virtualizace

S příchodem virtualizačních technik se naskýtá otázka, proč vlastně virtualizovat a jaké výhody virtualizace přináší. Důvodů, proč nasadit virtualizované řešení a proč je virtualizace v posledních letech tak populární, je více. Tom Bittman, analytik společnosti Gartner, říká: „*Virtualizace je v současnosti jedním z nejvlivnějších trendů v počítačovém světě. Dostupnost bezplatných hypervisorů bezpochyby zajistí růst trhu a nabídne také přesvědčivý argument pro společnosti, které dosud virtualizační procesy neimplementovaly. Výše uvedené platí zejména pro segment malého a středního businessu a rozvojové trhy. Samotný hypervisor je však pouhý základ. Obchodní model a skutečný přínos z virtualizace se vyvíjejí směrem k virtualizované infrastruktuře a managementu. Klíčové jsou také automatické nástroje znásobující efekt hypervisoru.*“ (vmwarenews.com, 2010 [19])

Původně se virtualizace vyvinula jako technika, která umožňovala běh několika logických systémů na jednom stroji. To přinášelo mnoho výhod. Pomineme-li lepší využití dostupného výpočetního výkonu (který byl v té době poměrně drahý), asi největší výhodou bylo, že vytvořením několika izolovaných prostředí se uživatelé stávali naprosto

nezávislými na ostatních a mohli tedy využívat „svůj“ operační prostor z hlediska uživatelských oprávnění na vyšší (administrátorské) úrovni. Zároveň se tím předcházelo různým konfliktům, které jinak v rámci víceuživatelských prostředí vznikaly (sdílení společných zdrojů, nedostatečná oprávnění k nějaké činnosti a další). V souvislosti s tím se také snížily administrátorské požadavky na údržbu takových systémů, jelikož v případě poruchy některého z virtualizovaných systémů došlo k pouhému nahrazení systémem novým. Zbývajících uživatelů se tato akce vůbec nedotkla a na jejich práci se to nijak neprojevilo.

Výše uvedené výhody jsou platné i dnes. Nižší administrátorské nároky (ať už se týkají samotných lidských zdrojů nebo využití technologických kapacit jako je například průchodnost sítě), izolace jednotlivých uživatelů a předcházení konfliktům jsou hlavními důvody pro nasazení virtualizace v podnicích. Techniky, kterými je toho dosaženo, se však postupem času zdokonalily a v mnoha oblastech se již používá technik nových (virtualizace aplikací). Stejně tak optimalizace využití hardwarového vybavení (a tím nižší ekonomické náklady) je obrovským přínosem, který ke konci minulého tisíciletí podpořil boom virtualizačních technik.

Tyto body však nepokrývají veškeré přínosy virtualizace. Svým izolovaným prostředím, které nenaruší okolní systémy, je virtualizace více než vhodná také pro vývoj a testování nových programů, především jader operačních systémů. Velmi často se také používá v případě serverů, kde se od sebe pomocí virtualizace oddělí jednotlivé serverové komponenty a díky tomu vznikne větší množství jednoúčelových serverů. Ty jsou v konečném důsledku mnohem stabilnější a bezpečnější. V případě pádu některého z těchto strojů pak dojde k nahrazení jeho bitovou kopií a celý proces obnovy je tak mnohem jednodušší. Stejně tak se zjednodušuje i případný upgrade některé ze serverových technologií.

S tím souvisí i další obrovská výhoda, kterou virtualizace přináší. To je vysoká přenositelnost celých systémů, a to nezřídka i mezi jednotlivými platformami. Díky tomu, že princip virtualizace je založen víceméně na abstrakci od jednotlivých hardwarových komponent a pracuje tedy v určitých případech se zjednodušeným modelem instrukcí, je jeho přenositelnost závislá v podstatě pouze na existenci vhodného hypervizoru na té které platformě. Tohoto faktu se využívá především při upgradu fyzického zařízení, kdy

přenositelnost virtuálního hosta podstatně zjednodušuje manipulaci a výrazně zkracuje dobu od začátku upgradu k nasazení. Ještě větší výhody pak migrace virtuálních strojů přináší v případě grid computingu, kdy se počet běžících fyzických strojů může dynamicky měnit v závislosti na spotřebě výkonu. V období menší zátěže se pak jednotlivé virtuální stroje mohou přesunout na několik málo fyzických strojů a tím šetřit náklady (opotřebení hardware, energetické zdroje a další).

**Grid computing** sice obsah této práce svou problematikou přesahuje, nelze se o něm ale nezmínit. Tato technologie umožňuje spojení více fyzických strojů (a tím i více výpočetního výkonu) pod jednu logickou entitu a s tou poté pracovat. Toho se využívá především u náročných výpočetních úkolů. V praxi jsou touto technikou realizovány především požadavky na vysoce spolehlivé a náročné aplikace, ke kterým přistupuje mnoho uživatelů, a zpracovávají tak velké množství dat. Jejich realizace konvenčními metodami by byla velmi obtížná (technicky i finančně) a v některých případech pravděpodobně i nemožná.

Pokud tedy shrneme výhody nasazení virtualizace, získáme následující body:

- Nižší náklady
- Lepší využití fyzických zdrojů (konsolidace serverů)
- Dynamické navyšování výkonu
- Vyšší bezpečnost
- Jednodušší migrace v rámci fyzických zařízení i platforem

Aby však nebyly uváděny pouze výhody, je nutné zmínit i některé nevýhody, které virtualizovaná řešení mají. Jedná se především o zvýšenou režii, kterou představuje provoz hypervizoru. Dále je pak nutné zmínit časové nároky na výpočty, které především z důvodu dělby zdrojů mezi několik logických systémů budou vyšší, než při provozu přímo na fyzickém hardwaru. Subjektivně se dá také říci, že nasazení virtuální infrastruktury bývá obtížnější, než klasický přístup. Toto je však později kompenzováno jednodušší správou a nižšími požadavky na údržbu.

Nevýhody virtualizace shrnuty v bodech jsou tedy následující:

- Vyšší režie

- Vyšší časové nároky na výpočet
- Obtížnější realizace
- Obtížné nasazení na nestandardizovaném hardware

## *3.3 Vývoj virtualizačních technik*

V posledních letech je pojem virtualizace často diskutován. Mluví se o jejích výhodách, úskalích a vhodnosti nasazení virtuálních technologií v tom kterém prostředí. Určitým paradoxem je, že samotný koncept této technologie je více než 40 let starý.

Jako první využila výhod virtualizace společnost IBM ve spolupráci s Cambridge Scientific Center, když v roce 1967 uvedla operační systém CP-40 (Control Program). Byl to logický důsledek vývoje v oblasti počítačové techniky, kterou tato zaznamenala v průběhu 60. let 20. století. Tato doba znamenala přechod od dávkového zpracování úloh k víceuživatelskému interaktivnímu prostředí. Většina společností v té době se rozhodla jít cestou time-sharing systémů, které však měly zásadní problémy se stabilitou a rychlostí. Společnost IBM však opustila ideu víceuživatelského prostředí a vypravila se cestou individuálních virtualizovaných operačních systémů, které běžely na jednom stroji. Tyto systémy se nazývaly CMS (Cambridge nebo Console Monitor System, později v operačním systému VM přejmenováno na Conversational Monitor System) a celý koncept pak CP/CMS. V následujících letech se pak tento koncept stal základem dominance společnosti IBM na poli mainframe počítačů.

V 80. a 90. letech se virtualizační techniky používaly pouze okrajově. Důvodem bylo opuštění konceptu klient-server a masivní rozšíření architektury x86, která byla cenově dostupná a dostatečně výkonná. Ke konci tisíciletí se však o virtualizaci opět začíná mluvit. Tím, jak se postupem času technologie x86 stala standardem i v oblasti serverů, rostl její výkon a klesala cena, bylo nasnadě začít využívat virtualizační technologie i v této oblasti. Leaderem na trhu se stala společnost VMware, která v roce 1999 představila technologii pro virtualizaci na architektuře x86.

## 3.4 Popek-Goldbergovy podmínky

Rámec virtualizace byl přesněji vymezen v roce 1974, kdy byl v magazínu Communications of the ACM publikován článek "Formal Requirements for Virtualizable Third Generation Architectures" pánů Geralda J. Popeka a Roberta P. Goldberga. V něm byly definovány základní podmínky, které musí plně vizualizovaný systém splňovat. I přesto, že od publikování článku uplynulo již mnoho let, jsou tyto podmínky stále relevantní a nejlépe vystihují rámec virtualizace. Důvodem je především to, že ačkoliv se technologie vyvíjejí čím dál tím rychleji, principy PC architektury se změnily pouze málo (například Von Neumannova architektura).

Tyto podmínky definují především vztah virtualizovaného systému k jeho hostiteli, ke zdrojům které využívá a způsob jak je využívá. Podmínky jsou celkem 3, ale pro zdůraznění podmínky izolace byly pro tuto práci rozpracovány do 4 okruhů.

**Izolace:** Každý virtualizovaný systém musí být plně izolován od ostatních virtualizovaných systémů, které využívají identických fyzických zdrojů.

Hlavním důvodem je zabezpečení toho, aby jakákoliv situace, která nastane u jednoho virtualizovaného systému, nenarušila chod ostatních virtuálních systémů. V praxi je toho docíleno existencí vrstvy mezi fyzickými zdroji a logickými systémy. Tato vrstva se nazývá hypervizor a věnovat se jí budeme dále.

**Správa zdrojů:** Hypervizor musí mít kompletní kontrolu nad přidělováním zdrojů jednotlivým virtuálním hostům.

Každému virtualizovanému systému je tedy přidělena množina zdrojů, které využívá pouze on a ke kterým nesmí mít, vyjma hypervizoru, přístup žádná jiná logická entita. Hypervizor musí zabránit přímému přístupu virtualizovaného systému k fyzické vrstvě a zajistit překlad požadavků virtualizovaného systému na fyzické zdroje. Důvodem tohoto překladu je izolace jednotlivých systémů. V případě, že by měl virtualizovaný systém přímý přístup k fyzickým zdrojům, mohl by svou činností ohrozit chod ostatních virtualizovaných systémů.

**Ekvivalence:** Program běžící ve virtualizovaném prostředí by měl vykazovat takové chování, které bude v podstatě totožné s tím, jak by se program choval v nevirtualizovaném prostředí.

Vyjma práce se zdroji tedy musí být veškeré procesy vykonávané ve virtualizovaném prostředí ekvivalentní k výkonu na fyzickém hardwaru. Výjimka zdrojů je způsobena skutečností, že se veškeré hostované systémy dělí o jedny zdroje. Hypervizor jim tedy zajišťuje pouze určité množství zdrojů (procesorového času, času k práci s periferiemi, …) a dá se tedy předpokládat, že tím budou vznikat určitá, především časová, omezení oproti běhu na fyzickém hardwaru.

Důsledkem, respektive průvodním jevem, tohoto pravidla je také skutečnost, že virtualizovaný systém by neměl vůbec tušit, že běží ve virtualizovaném prostředí.

**Efektivita:** Strojový kód, který je vykonáván virtualizovaným systémem, a který není v rozporu s předcházejícími pravidly, musí být vykonáván nativně, tedy bez zásahu hypervizoru.

Na rozdíl od emulace (viz kapitola 4.1), která provádí překlad všech instrukcí strojového kódu, by měla virtualizace provádět všechny operace nativně. Tato skutečnost samozřejmě nesmí odporovat výše uvedeným pravidlům a nesmí tedy ohrozit ostatní hostované systémy. Na rozdíl od emulace se tak docílí mnohem větší efektivity, jelikož se nemusí překládat všechny instrukce, ale překládají se pouze ty, které by mohly narušit systémové zdroje. V případě potřeby vykonat takovou instrukci, která je v rozporu s ostatními pravidly, musí její provádění převzít hypervizor.

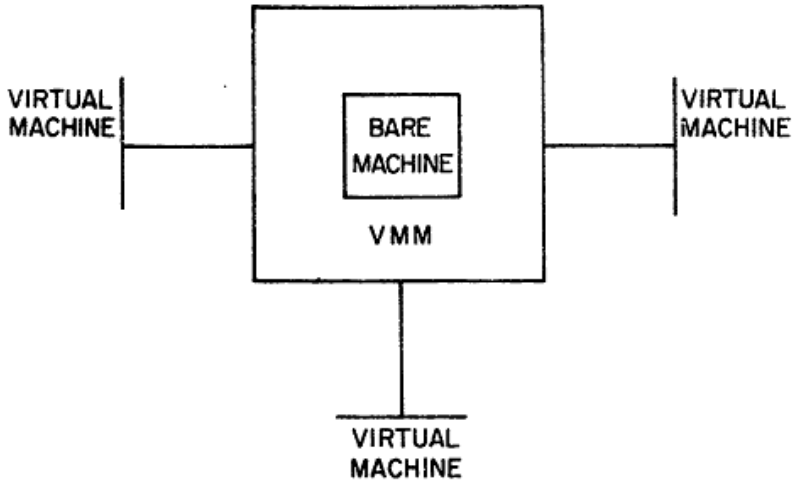
Tato pravidla, ve větší či menší míře, dodržují všechny typy virtualizace, o kterých pojednává tato práce. Jednotlivé způsoby virtualizace se liší především v tom, jakým způsobem splnění podmínek docilují. Ne všechny způsoby virtualizace však splňují všechna pravidla beze zbytku. Příkladem může být paravirtualizace, která již svou podstatou porušuje pravidlo o ekvivalenci (viz kapitola 4.3).

## 3.5 Hypervizor

Hypervizor (nebo též Virtual Machine Monitor či virtualizátor) je vrstva, která se většinou nachází mezi fyzickým hardwarem a virtualizovanými systémy. Většinou proto, že existují přístupy, ve kterých se využívá takzvaného privilegovaného operačního systému, který některé z funkcí hypervizoru přebírá (viz následující kapitoly). I v těchto případech však hypervizor existuje a jedná se o jedinou vrstvu, se kterou virtualizovaný systém komunikuje.

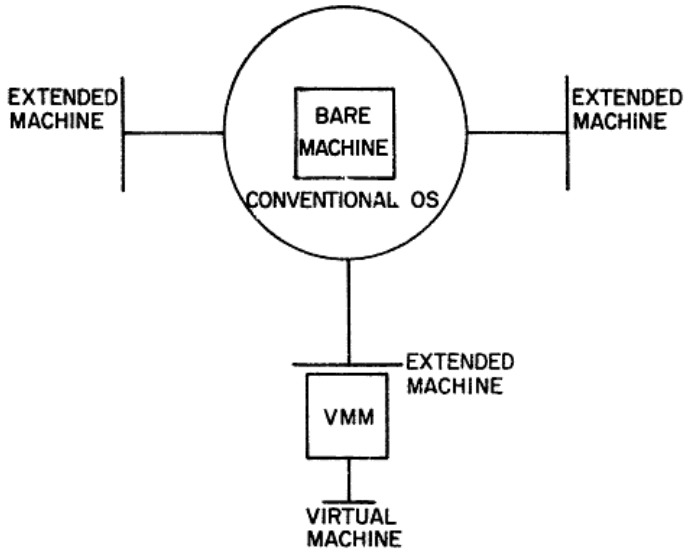
Hypervizor zajišťuje především izolaci virtualizovaných hostů od fyzického hardwaru. Zároveň však musí zajistit všechny funkce, které daný hardware poskytuje a to buď formou kontrolovaného povolení přístupu k němu, nebo formou překladu strojových instrukcí. Primárně se jedná o zajištění přístupu k procesoru, operační paměti a vstupně-výstupním zařízením. Důležitým úkolem, který také musí hypervizor plnit, je přidělování zdrojů a priorit jednotlivým virtuálním strojům.

Robert P. Goldberg definuje 2 základní typy hypervizoru. **Prvním typem** je hypervizor nativní, který je provozován přímo na konkrétním hardwaru. Tento typ hypervizoru se většinou označuje jako hostless. Jedná se operační systém, případně jádro operačního systému, který má v sobě implementovánu podporu virtualizace a je schopno zajistit plánování a alokaci zdrojů pro virtuální stroje. Problematická je však univerzálnost použití, jelikož tento hypervizor musí být napsán již s ohledem na skutečnost, že poběží přímo na hardwaru daného zařízení a musí tedy mít implementovány ovladače pro práci s daným fyzickým strojem. Klasickým představitelem tohoto modelu jsou mainframové počítače společnosti IBM, včetně počítače S/360 spolu s operačním systémem CP/CMS, který je považován za prvního představitele virtualizovaného řešení.




Obrázek č. 1 – Schematické vyjádření prvního typu hypervizoru – zdroj [20]

**Druhý typ** hypervizoru, nazývaný většinou hostovaný (hosted), běží v rámci klasického operačního systému a až pod ním běží hostované operační systémy. Výhodou je, že v tomto případě hypervizor zajišťuje pouze nezbytnou virtualizační vrstvu. Veškeré procesy alokace zdrojů a komunikace s periferiemi přenechává na hostujícím operačním systému. Problémem této realizace je zajištění privilegovaného módu pro běh hypervizoru, jelikož běží v aplikační úrovni. Řešení tohoto problému je základním problémem, před kterým virtualizace stojí.




Obrázek č. 2 – Schematické vyjádření druhého typu hypervizoru – zdroj [20]

Ani jeden z typů hypervizoru však nemůže běžet na jakémkoliv stroji. K jejich běhu je potřeba, aby především procesor splňoval některé náležitosti. Tyto náležitosti může splňovat buď nativně svým návrhem, nebo musí jejich splnění zajistit jinými (softwarovými) prostředky. Požadavky na procesor jsou tedy následující:

- *Metody interpretace neprivilegovaných instrukcí musí být zhruba stejné v privilegovaném i uživatelském módu. Procesor například nesmí u běhu v privilegovaném módu používat v instrukcích nebo přiděleném adresním prostoru additional bit.*
- *Musí existovat metody, například systémová ochrana nebo překlad adres, které zajistí ochranu hostujícího systému a ostatních virtuálních strojů před právě aktivním virtuálním strojem.*
- *Musí být zajištěn způsob, kterým je hypervizoru signalizován pokus virtuálního stroje o provedení kritické instrukce. Hypervizoru také musí být schopen emulovat průběh takové instrukce.* (Robin, Irvine, 2000, strana 4 [7])

Kritickými instrukcemi jsou:

- *Instrukce, které by mohly prozradit virtuálnímu stroji, že běží ve virtualizovaném módu nebo tento stav změnit.*
- *Instrukce, které přistupují k citlivým datům, jako jsou paměťové registry nebo registry přerušení.*
- *Instrukce, které odkazují na datová úložiště, paměťový systém nebo systém překladu paměti. Především se jedná o instrukce, které by mohly virtuálnímu stroji povolit překonat hranice jemu vymezeného prostoru.*
- *Veškeré vstupně-výstupní instrukce* (Robin, Irvine, 2000, strana 4 [7])

Tyto požadavky se týkají obou typů hypervizoru. Druhý typ hypervizoru však musí splňovat ještě některé dodatečné požadavky na hostující systém:

- *Hostující operační systém nesmí provádět žádnou činnost, která by mohla narušit podmínku stejné interpretace neprivilegovaných instrukcí v privilegovaném a uživatelském módu.*

- *Hostující operační systém musí mít schopnosti, které zaručí ochranu hypervizoru a virtualizovaných strojů před právě běžícím virtualizovaným strojem.* (Robin, Irvine, 2000, strana 4 [7])

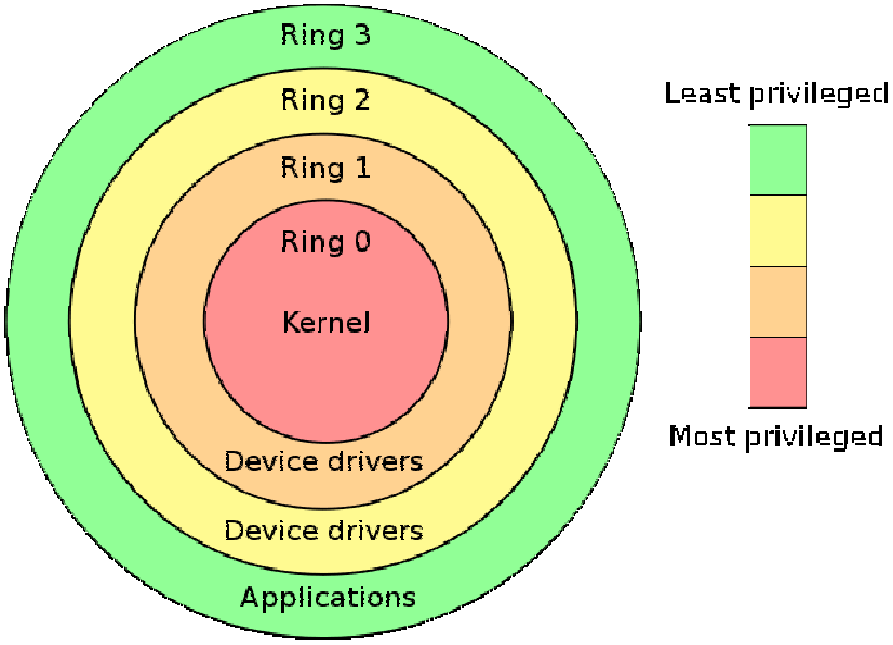
Velmi často se stává, že procesor nesplňuje některé podmínky pro provoz hypervizoru prvního nebo druhého typu. Pro tento případ byl zaveden pojem **hybridní hypervizor**. Ten se vyznačuje tím, že striktně nelpí na splnění všech požadavků a jejich splnění velmi často zajišťuje softwarovými prostředky (emulací instrukcí). Rozdíl tak spočívá především v tom, že hybridní hypervizor překládá veškeré privilegované instrukce, zatímco hypervizor prvního nebo druhého typu umožňuje přímou interpretaci nekritických privilegovaných instrukcí. V dalším textu bude používán pouze termín hypervizor, bez rozlišení jednotlivých typů. Toto rozlišení většinou vyplyne z kontextu fungování jednotlivé metody. Pouze tam kde to nebude z popisu funkcionality zřejmé, bude explicitně označen typ hypervizoru.

## 3.6 Virtualizace procesoru

Jak již bylo nastíněno v části týkající se hypervizoru, základní problematika virtualizace se týká přidělování fyzických zdrojů jednotlivým virtualizovaným systémům. Jedním z podstatných zdrojů je procesorový čas, na kterém závisí doba potřebných výpočtů a samotný běh hostovaného systému vůbec. Existuje několik způsobů, jak rozdělit potřeby jednotlivých systémů mezi kapacitu procesoru. Tyto metody se liší v případě úplné virtualizace, paravirtualizace, hardwarové virtualizace a dalších (viz kapitola 4).

Pro pochopení, jakým způsobem hypervizor vůbec může postupovat v případě přidělování procesorového času jednotlivým virtualizovaným systémům, je důležité pochopit metody ochrany, které současné procesory používají. Většina dnešních sofistikovaných procesorů používá ochranu pomocí privilegovaného módu a uživatelského módu. V rámci privilegovaného režimu, který má oprávnění k provádění kritických instrukcí strojového kódu, které mohou ohrozit chod počítače, běží všechny kritické procesy, jako jsou jádro operačního systému, procesy zajišťující komunikaci se vstupně-výstupními zařízeními, mapování paměti a další. V uživatelském režimu pak běží všechny ostatní aplikace, které svou činností nemohou ohrozit chod počítače. V případě že proces, který běží v uživatelském režimu (respektive obecně ve vrstvě s menším oprávněním),

potřebuje vykonat nějakou z privilegovaných instrukcí strojového kódu, musí použít systémové volání a zajistit si pomocí některého z procesů běžících v privilegovaném módu oprávnění k provedení těchto instrukcí.




Obrázek č. 3 – Schematické vyjádření ochrany procesoru pomocí metody Ring – zdroj www.wikipedia.org

Konstrukce virtualizace je poměrně jednoduchá v případě, že veškeré instrukce instrukční sady, které nějakým způsobem ovlivňují stav systémových prostředků (paměť, I/O zařízení, …) nebo jsou na tomto stavu závislé, spadají do množiny privilegovaných instrukcí. V případě architektury x86 (potažmo x86_64, ale i jiných) je problémem právě existence instrukcí, které svou činností nebo svými potřebami, ovlivňují stav těchto prostředků a přitom běží v uživatelském módu. Tento problém je závažný především proto, že neexistuje jednoduchý způsob jak zajistit zjištění těchto instrukcí hypervizorem. V případě, že by tyto instrukce spadaly do množiny privilegovaných instrukcí, bylo by jejich zjištění při běhu v uživatelském módu zajištěno vyvoláním přerušení. Tyto instrukce však svou činnost v uživatelském módu nijak nedeklarují a jejich zjištění je tedy poměrně komplikovanou záležitostí. Věnovat se jí budou kapitoly o jednotlivých metodách virtualizace.

Ošetření existence obtížně identifikovatelných konfliktních instrukcí je ovšem pouze jedním problémem při využívání procesorů k virtualizaci. Dalším problémem je zajištění

ochrany samotného hypervizoru, ideálně zapouzdřením do vlastního ochranného módu. V praxi je pohled na dvojici privilegované a uživatelské úrovně vidět u architektury x86. Většina novějších procesorů na bázi x86_64 v sobě obsahuje více než 2 úrovně ochrany, kdy nejvyšší (v které běží jádro operačního systému) používá označení Ring 0 a nejnižší (uživatelský mód) většinou označení Ring 3. Mezi nimi se nacházejí „ringy“, v kterých běží většinou ovladače vstupně-výstupních zařízení a další důležité moduly.

Problematika virtualizace procesorové kapacity se mimo jiné zabývá skutečností, jak zajistit vznik další vrstvy ochrany, ve které poběží hypervizor. V zásadě existují 2, respektive 3, způsoby jak toho docílit. Každý z nich má však svoje úskalí, která se musí při realizaci brát v potaz.

**Ring deprivileging**

Jedná se o techniku, ve které se privilegovaný mód virtualizovaného operačního systému „poníží“ na ring, jehož ochrana je o stupeň nižší (z Ring 0 na Ring 1). Tento způsob je realizovatelný pouze v případě, že architektura podporuje více než 2 stupně ochrany. Nelze tedy použít například u architektury x86. Dále je potřeba, aby byla hardwarově zajištěna nadřazenost Ring 0 nad Ring 1. Pokud by toto zajištěno nebylo, mohlo by docházet ke konfliktům, při kterých by se činnost operačního systému mohla prolínat s činností hypervizoru. V takovém případě by byla porušena Popek-Goldbergova definice virtualizace.

**Ring compression**

Ve své podstatě se jedná o techniku Ring deprivilegingu, která je ovšem aplikovaná na počítačových architekturách, které umožňují pouze 2 stupně ochrany – privilegovaný a uživatelský mód (procesor není schopen rozlišit módy 0-2). V tomto případě, v rámci zajištění ochrany hypervizoru, běží hostovaný operační systém společně s aplikacemi v Ring 3 a samotný hypervizor běží v privilegovaném módu Ring 0. Problémem u této metody tak není ochrana hypervizoru, která je zajištěná během v privilegovaném módu, ale ochrana hostovaného operačního systému před jeho vlastními aplikacemi, jelikož tyto běží společně s ním v Ring 3.

Implementace tohoto řešení je tak velmi obtížná. To je důvodem toho, že se první způsoby virtualizace architektury x86 objevily až v průběhu 90. let 20. století.

**Hyperprivileging**

Poslední metoda se od předchozích 2 principielně velmi odlišuje. Zachovává původní rozložení ochrany a pro vrstvu hypervizoru zavádí úplně novou vrstvu ochrany, označovanou jako Ring -1. Tento postup však musí být podporován hardwarem, nicméně oba hlavní výrobci procesorů pro osobní počítače, tedy společnosti Intel a AMD, tuto technologie do svých procesorů již nasazují. V případě Intelu se tato technologie označuje Intel VT-x (kódové označení Vanderpool), v případě AMD se označuje AMD-V (kódové označení Pacifica).

U všech metod zajištění ochranné vrstvy pro hypervizor je důležité vytvořit takové podmínky, aby hostovaný operační systém nedetekoval, že běží v jiném než privilegovaném módu a zároveň aby se hypervizor dozvěděl o všech systémových voláních. Toto ošetření běhu operačního systému v neprivilegovaném módu se týká především některých instrukcí strojového kódu, jako jsou PUSH, STR nebo MOV, a které musí být hypervizorem zachyceny, rekompilovány a provedeny.

## 3.7 Virtualizace paměti

Problematika virtualizace paměti se týká především definice modelu správy paměti, který je schopen zajistit překlad paměťové adresy z logického prostoru virtuálního stroje na paměťovou adresu stroje fyzického. Navíc musí zajistit, že bude fyzický adresový prostor v souladu s požadavky rozdělen mezi hypervizor a jednotlivé virtualizované stroje, které budou mít přístup pouze k jim přiděleným segmentům.

Princip tohoto překladu by se dal připodobnit k vytváření logických segmentů pro jednotlivé aplikace v rámci jednoho operačního systému. Jádro operačního systému vytváří logické bloky paměťového prostoru, které dle potřeby přiřazuje jednotlivým procesům. Jednotlivé procesy adresují svá data do těchto logických bloků a teprve jádro operačního systému je poté adresuje do prostoru fyzické paměti. Jádro operačního systému tak musí udržovat nejen informace o překladu dat z logické do fyzické roviny, ale musí znát informace o tom, jakému procesu data patří, jaká jsou oprávnění ke čtení/zápisu, zda se

obsah paměti změnil (tzv. dirty bit) a další. Musí také zajistit izolaci dat jednotlivých procesů, aby nedošlo ke konfliktům a byla zajištěna bezpečnost zpracovávaných dat. K tomu všemu využívá různých technik, které však přesahují rámec této práce.

Princip přidělování adresového prostoru jednotlivým virtualizovaným strojům probíhá v obecné rovině stejným způsobem. Jednotlivé způsoby virtualizace se ale liší v konkrétní realizaci. O tom, jakým způsobem hypervizor konkrétně přiděluje paměť, bude pojednáno v kapitole 4, zabývající se typologií virtualizačních řešení.

## *3.8 Virtualizace vstupně-výstupních zařízení*

V obecné rovině je opět nezbytné, aby veškerý přístup k periferiím mohl kontrolovat hypervizor a v případě potřeby veškerou komunikaci odchytil a předcházel případným kolizím. Oproti virtualizaci procesorů a pamětí je tato oblast komplikována skutečností, že existuje nepřeberné množství různého periferního hardwaru, který často komunikuje s interními částmi počítače rozdílnými způsoby.

Na fyzické úrovni mohou vstupně-výstupní zařízení komunikovat se systémem v zásadě třemi způsoby.

Prvním je **programové řízení přídavných zařízení**. V tomto případě procesor řídí celý proces přenosu dat do/ze vstupně-výstupního zařízení. Tato zařízení jsou většinou namapována na určité segmenty operační paměti a procesor k nim přistupuje pomocí stejných instrukcí, jako k operační paměti. Na některých platformách je však pro I/O zařízení vyhrazen vlastní adresní prostor a v rámci strojového kódu existují instrukce pro čtení/zápis z tohoto prostoru. Problémem této technologie je značné plýtvání procesorovým časem.

Proto vznikl druhý přístup ke komunikaci periferních zařízení se systémem. Ve své podstatě vychází z prvního, ale je doplněn o prvek **přerušení**, které zajišťuje součinnost s procesorem pouze ve chvíli, kdy je celý přenos připraven a je vyžadována součinnost s procesorem. Ten se tak v mezičase může věnovat jiným procesům a nečeká na zařízení. I přes tento inovativní prvek však celou komunikaci zajišťuje procesor.

Třetí metoda je **přímý přístup k paměti** (Direct Memory Access – DMA), kdy přenos dat do paměti řeší samotná periferie. Procesor pouze vydá svolení s tímto přenosem a informacemi o počáteční adrese zápisu, délce bloku, směru přenosu a dalších. Po skončení přenosu pak zařízení pomocí přerušení informuje procesor o dokončení tohoto procesu. Tato technika vykazuje mnohem větší efektivitu než metoda programového řízení, ale má několik omezujících limitů. Prvním z nich je šířka sběrnice, která limituje maximální možnou délku přenosu. V případě přenosu většího množství dat je tak potřeba žádat procesor o svolení a informace o cílové destinaci v paměti několikrát. Dalším problémem je pak pouze omezené množství přerušení, kterými se jednotlivá zařízení hlásí procesoru. Kvůli tomu je omezeno množství zařízení, která tímto způsobem mohou komunikovat.

Virtualizace periferních zařízení tedy klade následující problémy. Prvním z nich je, jak už bylo napsáno výše, odchycení komunikace zařízení se systémem. Druhým je samotná identifikace zařízení pro virtualizovaný systém. V konvenčním prostředí běžných operačních systémů je se zařízením komunikováno a zařízení je identifikováno pomocí ovladačů. V praxi je velmi obtížné zajistit virtualizaci všech ovladačů jejich modifikací. Toho se v praxi využívá pouze v případě paravirtualizace (viz. kapitola 4.3). Z toho důvodu většinou hypervizor vytváří virtuální zařízení, jež dává k dispozici hostovanému systému. Toto zařízení je většinou identifikováno jako zařízení obecného typu a předpokládá se, že jeho ovladače jsou již implementovány v používaném operačním systému. Hypervizor pak zajišťuje překlad mezi tímto virtuálním zařízením a fyzickou periferií. Do třetice je to identifikace zařízení hypervizorem. I ten musí s fyzickým zařízením nějak komunikovat. V praxi se využívají dvě metody. První spočívá ve vytvoření ovladačů pro samotný hypervizor. Tento způsob je sice velmi efektivní, ale naráží opět na problém existence mnoha rozličných zařízení a nutnosti vytvoření ovladačů, které budou vhodné pro hypervizor. Elegantnějším řešením je využití existence ovladačů pro některý z virtualizovaných systémů. Hypervizor pak komunikuje s těmito vstupně-výstupními zařízeními pomocí tohoto virtualizovaného systému.

V některých případech je vhodné vyčlenit samotné fyzické zařízení pouze pro virtualizovaný systém. Tím odpadá starost s virtualizací, jelikož k tomuto zařízení má výhradní přístup hostovaný operační systém a ostatní virtualizované systémy o jeho

existenci vůbec neví. Virtualizovaný stroj si pak řídí veškerou komunikaci se zařízením, ale nemůže tím nijak narušit chod okolních systémů. Často se toho využívá v případě síťových karet, jelikož se tím značně zrychlí komunikace po sítí a administrace sítě jako celek. Obtížněji implementovatelná, ale v poslední době v praxi velmi často využívaná, je technika přidělení jednotlivých fyzických jader procesoru jednotlivým virtuálním entitám.

# 4 Komparace virtualizačních technik

## *4.1 Emulace*

Vzhledem k tomu, že cíle a principy emulace se od virtualizačních řešení v mnohém odlišují, nebude se emulací tato práce podrobněji zabývat. Nezmínit ji by však byla chyba, jelikož ve své podstatě se může jednat o jednu z hraničních forem virtualizace.

Pojem emulace bývá velmi často zaměňován s pojmem simulace. V některých ohledech se tyto dva pojmy skutečně dají zaměnit a hranice mezi nimi je velmi tenká. V pojetích, jak by tyto pojmy měly být chápány je však velký rozdíl. Simulace modeluje procesy, o jejichž fungování máme jen málo informací. Jejím hlavním účelem je tedy shromáždění dodatečných informací, s využitím modelování potřebných jevů. Na druhou stranu v případě emulace nějakého systému, máme o jeho fungování velmi dobré informace, ale z nějakého důvodu musíme jeho funkci nahradit.

„*Rozdíl mezi simulací a emulací je tedy především v tom, k čemu slouží - simulace k získání nových poznatků o určitém systému, zatímco emulace umožňuje zajištění jeho funkcí jinými prostředky.*“ (Peterka, 2010 [17])

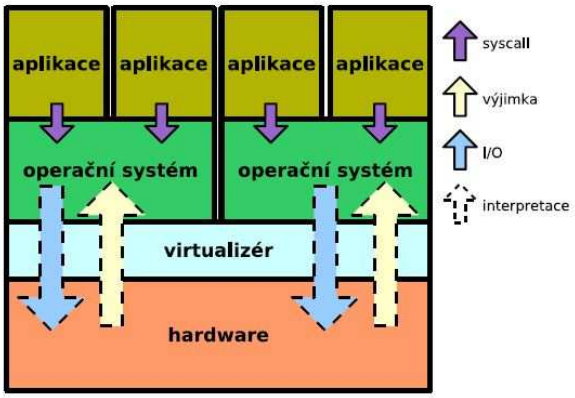
Z toho je patrné, jaká je souvislost emulace s virtualizací. Virtualizace se také snaží „emulovat“ nějakou platformu, na které by mohl být provozován virtualizovaný systém. Na rozdíl od virtualizace, která se z důvodu efektivity snaží pouze o nezbytné zásahy do strojových instrukcí hostovaného stroje, emulace se snaží dosáhnout svou činností iluzi dokonalé kopie často úplně odlišné platformy. Této skutečnosti je docíleno úplným překladem všech strojových instrukcí a veškerých požadavků z instrukční sady hostovaného systému do instrukční sady hostujícího systému. Tento překlad je však velmi často realizován za cenu vysokých režijních nákladů, které v mnoha případech znamenají

razantní snížení výkonu. Nezřídka se v případě emulace používají kvůli abstrakci vyšší programovací jazyky (C/C++) a režie se tak zvyšuje ještě o interpretaci těchto jazyků. Technika emulace tak není většinou vhodná k hostování logických systémů. To ovšem nijak nesnižuje její použitelnost a výhody, které poskytuje. Jako příklad může posloužit třeba takzvaná cross-compilation, kdy se provádí kompilace modulů jádra operačního systému, či jádra samotného, pro určitou platformu na platformě jiné (většinou výkonnější).

## 4.2 Úplná virtualizace

V porovnání s emulací se úplná virtualizace snaží o mnohem efektivnější chod hostovaných systémů. Toho je docíleno především faktem, že provádění většiny instrukcí není potřeba emulovat. Virtuální stroj používá stejnou sadu instrukcí, jakou disponuje hostující zařízení. Z toho plyne již výše zmíněná výhoda. Instrukce, které nejsou kritické (nemění stav nebo nejsou závislé na systémových prostředcích a běží v privilegovaném módu) jsou prováděny nativně a tudíž efektivněji (rychleji). Na druhou stranu to znamená jisté omezení, jelikož virtualizovaný software musí být schopen nativního běhu na stejném hardwaru, na kterém je virtualizován.

Úplná, nebo také plná či nativní, virtualizace musí splňovat všechny Popek-Goldbergovy podmínky, nebo nějakým způsobem jejich splnění zajistit. Toho se dá docílit buď softwarovými prostředky, které budou uvedeny záhy, nebo hardwarovými prostředky, které budou popsány v kapitole o hardwarové vizualizaci (kapitola 4.4).




Obrázek č. 4 – Schéma úplné virtualizace – zdroj Děcký [8]

### 4.2.1 Virtualizace procesoru

V předchozích kapitolách byly definovány podmínky, které jsou kladeny na hypervizor k zajištění virtualizace. Tyto požadavky byly podmíněny určitými nároky na procesor (viz kapitola 3.5). Bohužel ne všechny platformy tyto podmínky splňují. Plně virtualizovatelné jsou například architektury sun4v od společnosti SUN Microsystems nebo architektura POWER (respektive PowerPC) od společnosti IBM. V následujícím textu se budeme zabývat především architekturou x86 (x86_64). Většina podmínek virtualizace je dnes již automaticky zahrnována do návrhu procesoru (více režimů ochrany, metody dynamické relokace paměti, systém asynchronních přerušení). Jejich splnění je totiž důležité nejen z důvodu potenciální virtualizace, ale je vyžadováno i pro běh například multitaskingových operačních systémů. Problém bývá v případě ošetření kritických instrukcí, které mohou měnit konfiguraci systémových zdrojů, nebo je touto konfigurací ovlivněno jejich chování, ale které nespadají do množiny privilegovaných instrukcí. Veškeré privilegované instrukce při každém pokusu o spuštění v uživatelském módu vyvolají přerušení a hypervizor se tak o jejich činnosti dozví a může korigovat jejich provedení. Kritické instrukce však v uživatelském módu žádné přerušení nevyvolají a hypervizor tedy musí zajistit jejich zjištění jinou cestou.

V případě architektury x86 se jedná především o tuto množinu instrukcí:

- POPF
- POP, CALL, JMP, INT, RET
- MOV

Tyto instrukce v uživatelském režimu nevyvolávají výjimku, ale jejich provedení se v uživatelském a privilegovaném režimu značně odlišuje. Je to způsobeno buď tím, že v uživatelském režimu ignorují některé příznaky (POPF), v případě přerušení a návratu vlivem skoku dochází ke změně zásobníků (POP, CALL, JMP, INT, RET) nebo vlivem odlišného způsobu testování ochrany paměti v privilegovaném a uživatelském módu (MOV). Mimo této skupiny instrukcí se jedná ještě o instrukce, které svou činností narušují podmínku ekvivalence, jak ji definovali Popek a Goldberg (viz. kapitola 3.4).

- SGDT, SIDT, SLDT

- SMSW, PUSHF
- LAR, LSL, VERR, VERW
- PUSH
- STR
- MOV

Tyto instrukce umožňují na základě rozdílu mezi hodnotami fyzického a virtuálního stroje nebo na základě odlišných hodnot pro privilegovaný a uživatelský mód identifikovat běh systému ve virtuálním režimu.

Možnosti virtualizace platformy x86 se zlepšily s příchodem architektury x86_64. Avšak i tato architektura obsahuje nejméně 3 kritické instrukce, které porušují některou z podmínek. Za předpokladu, že není možné ošetřit výskyt kritických instrukcí pomocí volání hypervizoru, musí být použita technika, která umožní odhalení těchto instrukcí ještě před jejich provedením. Všechny dále popsané techniky tak logicky vychází z analýzy proudu instrukcí ještě před jejich provedením. Kritické instrukce jsou v průběhu analýzy ošetřeny breakpointy, které zajistí volání hypervizoru a správnou interpretaci následné instrukce. V zásadě existují 3 techniky, kterými lze toto provést.

**Emulace kritických instrukcí**

Stejně jako v případě emulace je tato metoda založená na interpretaci jednotlivých instrukcí, které požaduje provést virtualizovaný systém. Oproti emulaci je však v případě úplné virtualizace využito faktu, že virtualizovaný systém používá stejnou sadu instrukcí jako hostující zařízení. Z toho důvodu není nutné interpretovat všechny instrukce, ale pouze ty kritické. Nekritické instrukce jsou prováděny nativně. Oproti emulaci to znamená výrazný výkonnostní vzestup a podstatné zvýšení efektivity.

Problematické je však stále postupné procházení instrukcí, kdy o každé instrukci interpret (hypervizor) rozhoduje, zda je či není nutná její emulace. Neefektivita tohoto řešení je výrazná především v případě mnohonásobně používaných bloků stejných instrukcí, kdy při každém volání každé instrukce je rozhodováno o emulaci či neemulaci.

### Statický překlad

Výše zmíněnou nevýhodu emulace kritických instrukcí se snaží odstranit metoda statického překladu. V tomto případě hypervizor nejprve projde celou posloupnost instrukcí a kritické instrukce nahradí ekvivalentními nekritickými instrukcemi. V případě, že z nějakého důvodu není možná záměna těchto instrukcí v době analýzy, vloží před ní hypervizor breakpoint, který ho zavolá v době zpracovávání instrukce za běhu a instrukci interpretuje později.

Tato metoda se však potýká s několika úskalími. Vzhledem k tomu, že dochází ke změně posloupnosti instrukcí ještě před jejich spuštěním, hrozí porušení podmínky ekvivalence dle Popek-Goldberga. To lze obejít zkopírováním instrukcí na jiné místo v paměťovém registru, bohužel tento prvek pak opět značně snižuje efektivitu. Mezi další problémy lze zařadit proměnnou délku instrukcí u některých architektur (CISC), kdy je jen obtížně zjistitelné, kde končí jedna instrukce a začíná další, a nepřímé skoky, u kterých je takřka nemožné zjištění cílové destinace dříve než při zpracování instrukce.

### Dynamický překlad

Problémy spojené se statickým překladem a emulací instrukcí se snaží efektivně řešit dynamický překlad. Stejně jako v případě emulace kritických instrukcí je použito dynamického překladu za běhu. Na rozdíl od emulace, však stejně jako statický překlad, interpretuje celé bloky instrukcí. To přináší výhodu především v opakovaném provádění některých bloků instrukcí. Pokud již byly jednou přeloženy, není nutné je znovu interpretovat a lze je rovnou provést. Tato metoda překladu je nejrozšířenější a víceméně veškeré moderní virtualizační produkty ji používají.

Aby bylo možné interpretovat celý blok instrukcí, musí blok obsahovat pouze instrukce, které nenaruší sekvenční zpracování programu. Nesmí tedy obsahovat instrukce skoku nebo větvení, které sekvenci narušují. Tyto instrukce, spolu s kritickými instrukcemi, je nutné nějakým způsobem identifikovat a zajistit vyvolání hypervizoru k jejich interpretaci. K tomu hypervizor používá procesorový

Debug Register. Do něj v průběhu interpretace ukládá adresy všech instrukcí skoku, větvení nebo kritických. Díky tomu je zajištěno, že v případě zpracovávání těchto instrukcí procesorem dojde, na základě porovnání s adresami v Program Counter registru, k vyvolání hypervizoru. Ten v případě výskytu kritické instrukce provede její emulaci. V případě výskytu instrukce skoku nebo větvení, provede zjištění, zda byl odkazovaný kód již analyzován a případně provede jeho analýzu a následné spuštění.

Z hlediska hypervizoru je důležité zajistit nemodifikovatelnost již jednou zkontrolovaných bloků. V případě nezajištění této skutečnosti by nebylo možné spouštět již zkontrolované bloky bez opětovné kontroly a jednalo by se tak prakticky o emulaci. K zajištění této nemodifikovatelnosti je používáno pro části paměti se zkontrolovanými bloky nastavení pouze pro čtení. Jakýkoliv pokus o zápis do těchto částí paměti tak vyvolá hypervizor, který má možnost provést analýzu modifikovaného kódu. Bohužel takto zjednodušený postup by mohl narušit podmínku ekvivalence, jelikož virtualizovaný systém by mohl přečtením obsahu paměti zjistit modifikaci kódu. Z toho důvodu se využívá duplicita těchto kódů, kdy provádění instrukcí je zajišťováno z jiného paměťového umístění, než kam je zapisován hostovaným operačním systémem. Tato skutečnost však zvedá režijní náklady hypervizoru, s čímž je potřeba počítat.

Předchozí odstavce jsou však pouze popisem obecných principů, které se v případě dynamického překladu instrukcí používají a na kterých je tento překlad založen. Konkrétní implementace dynamického překladu u jednotlivých virtualizačních produktů se v konečném důsledku velmi odlišují. Jejich cílem je totiž zajistit minimalizaci volání hypervizoru. Tato optimalizace pak zajistí mnohem efektivnější a rychlejší zpracování. V mnoha případech je však zajištění této skutečnosti součástí proprietálních řešení a informace o nich nejsou veřejně dostupné.

### 4.2.2 Virtualizace paměti

Jak již bylo uvedeno v kapitole o virtualizaci paměti (kapitola 3.7), základním problémem práce s virtualizovanou paměťovou jednotkou je zajištění mechanismů, které umožní překlad logické paměťové adresy virtuálního stroje na paměťovou adresu stroje

fyzického. Navíc je třeba omezit virtualizovaný systém na používání pouze těch částí paměti, které jsou mu přiděleny hypervizorem a zakázat přístup k ostatním částem fyzické paměti. Toho se dociluje obdobně, jako v případě správy paměti multitaskingovými operačními systémy. Takový paměťový model obsahuje stránky virtuální paměti o určité velikosti a přidělené jednotlivým procesům, které jsou překládány do fyzické paměti. Tabulky stránek ve fyzické paměti jsou pak pomocí metod ochrany paměti přístupné pouze pro jádro operačního systému. Tím je zajištěna ochrana jednotlivých paměťových prostor před ostatními procesy.

V rámci virtualizace paměťových jednotek se využívá existence primární tabulky stránek každého hostovaného systému, která obsahuje informace o tabulkách stránek jednotlivých procesů. Celý model se pak posune v podstatě o úroveň výš. Hypervizor vytvoří stínovou primární tabulku stránek pro každý systém, kterou udržuje ve stejném stavu jako je primární stránka hostovaného systému, ale je uložená ve fyzické paměti. Největším problémem je pak zajištění konzistence mezi těmito dvěma stránkami a identifikace případných změn ze strany hostovaného operačního systému. K zajištění konzistence a identifikaci změn lze principielně použít 2 způsoby. Prvním je identifikace změny v případě zápisu do primární tabulky stránek. Tato identifikace je realizována nastavením tabulky pouze pro čtení. Jakýkoliv pokus o zápis pak vyvolá přerušení, které odchytí hypervizor a provede zápis a synchronizaci. Druhou metodou je odchycení změn při pokusu operačního systému o načtení záznamu. Za předpokladu, že je překlad adres prováděn až při pokusu o načtení, neobjevuje se ve stínové tabulce v době tohoto pokusu. To vyvolá přerušení, při kterém hypervizor zajistí překlad a opětovné provedení instrukce čtení z tabulky.

Problémem obou výše zmíněných přístupů je vysoká režie hypervizoru, která je způsobena jeho častým voláním. Bohužel v případě úplné virtualizace ještě nebylo nalezeno efektivnější řešení (na rozdíl například od paravirtualizace – viz kapitola 4.3).

### 4.2.3 Virtualizace vstupně-výstupních zařízení

Největší překážkou ve virtualizaci vstupně-výstupních zařízení je, jak již bylo uvedeno, jejich množství. Není tedy v silách vývojářů zajistit spolupráci se všemi periferiemi. Proto se v rámci virtualizovaných systémů používají především obecnější

zařízení bez specializovaných funkcí. Výjimku mohou tvořit zařízení, která jsou ve své fyzické podobě vyhrazena pro použití v jediném virtuálním systému. K těmto zařízením pak má výhradní přístup pouze jediný virtualizovaný systém a díky tomu, že není nutné interpretovat jeho požadavky, může využít veškerou paletu instrukcí, které zařízení podporuje.

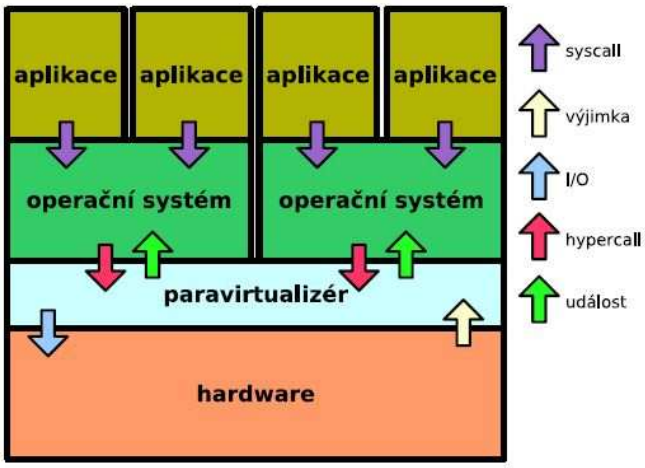
V rámci sdíleného přístupu více hostovaným systémů k periferiím je však potřeba zajistit překlad požadavků virtualizovaných systémů na fyzické zařízení a zároveň zajistit přenos dat z fyzického zařízení ke správnému virtualizovanému stroji. K této činnosti se využívá emulace zařízení. V rámci hostovaného systému se pak toto emulované zařízení hlásí jako přesná kopie některého z obecných fyzických zařízení. Díky tomu většinou není problém s ovladači a komunikací mezi virtualizovaným systémem a takto emulovaným zařízením vůbec. Hypervizor se pak stará o překlad instrukcí emulovaného zařízení na konkrétní fyzickou periferii a zajištění přístupu všech virtualizovaných entit k této periferii. Prakticky je volání hypervizoru, při pokusu o přístup k emulovanému I/O zařízení, realizováno opět pomocí přerušení. Stejně tak opačný proces, kdy komunikaci vyžaduje fyzické zařízení. Hypervizor akorát přesměruje datový proud z fyzického zařízení do patřičného paměťového prostoru virtuálního stroje.

Problémem emulace zařízení, jak již bylo zmíněno v jiné souvislosti s emulací, je její vysoká náročnost na výpočetní výkon. Naopak její výhodou, v případě vhodně navržené implementace, může být její vysoká přenositelnost mezi různým fyzickým hardwarem. Vzhledem k tomu, že stav emulovaných zařízení je uložen výhradně v paměti, je jedinou podmínkou pro migraci virtuálního stroje využívajícího emulovanou periferii pouze požadavek na znalost tohoto emulovaného vstupně-výstupního zařízení hypervizorem.

## 4.3 Paravirtualizace

Splnění všech podmínek virtualizace, tak jak je definovali Popek a Goldberg, je v případě mnoha architektur obtížný úkol. Úplná virtualizace se nedostatky jednotlivých architektur snaží zajistit výše popsanými, převážně softwarovými a poměrně náročně implementovatelnými, metodami. Proti tomu stojí paravirtualizace, která vědomě některé z podmínek porušuje.

Principielně je paravirtualizace velmi podobná virtualizaci úplné. Hostovaný operační systém musí být schopen využívat stejnou instrukční sadu jako fyzický hardware, přičemž veškeré nekritické instrukce by měly být prováděny nativně. Podstatný rozdíl, který mezi oběma přístupy existuje, je faktické a vědomé porušení podmínky o ekvivalenci (respektive izolaci, jak byla definována v kapitole 3.4) ze strany paravirtualizovaných systémů. Tato skutečnost je umožněna modifikací samotného paravirtualizovaného systému, který je upraven tak, aby o běhu ve virtuálním prostředí věděl a s tímto vědomím při svém chodu počítal. Výhody tohoto řešení jsou zřejmé. Není třeba složitě ošetřovat zjištění kritických instrukcí, jelikož paravirtualizovaný systém se jim buď vyhne, nebo na jejich existenci hypervizor sám upozorní. Stejně tak není potřeba řešit ochranu paměti, jelikož sám paravirtualizovaný systém respektuje vzniklá omezení, která pro něj z běhu v paravirtualizovaném prostředí plynou. Veškeré tyto, a samozřejmě i další, výhody paravirtualizace značně snižují režii na provoz hypervizoru a zefektivňují provoz paravirtualizovaných systémů. To všechno za podmínky, že daný paravirtualizovaný systém splňuje bezpodmínečně požadavek na správu zdrojů výhradně prostřednictvím hypervizoru. V opačném případě by paravirtualizace vedla k výskytu konfliktů a chyb, které by pravděpodobně způsobily kolaps všech paravirtualizovaných systémů běžících na daném fyzickém zařízení. Největším problémem paravirtualizace bývá úprava hostovaných systémů pro běh v paravirtualizovaném prostředí. V případě systémů s otevřeným kódem, není modifikace příliš složitá, jelikož většina moderních operačních systémů je nativně připravena pro běh v různých prostředích a na různých platformách. Problematická je tak pouze paravirtualizace proprietálních uzavřených systémů, které je buď velmi obtížné nebo často i nemožné modifikovat pro účely paravirtualizace. Z toho důvodu se paravirtualizace využívá poměrně často v případě periferií. V takové situaci stačí pouze vhodně upravit ovladače příslušného vstupně-výstupního zařízení a paměť a procesor virtualizovat metodami úplné virtualizace.




Obrázek č. 5 – Schéma paravirtualizace – zdroj Děcký [8]

### 4.3.1 Paravirtualizace procesoru

Stejně jako v ostatních metodách virtualizace procesoru se v případě paravirtualizace řeší především nedostatky původních návrhů jednotlivých architektur. Jedná se především o ošetření kritických instrukcí, které svou podstatou mohou narušit spolehlivost a stabilitu běžících systémů. Oproti emulaci a úplné virtualizaci, které řeší nedostatky počítačových architektur ex-post a proces virtualizace je tím ztížen o nutnost identifikace běhu kritických instrukcí, metoda paravirtualizace svou podstatou proces virtualizace předvídá a díky tomu je schopna na tyto instrukce upozornit nebo se jejich používání vyhnout. Již v případě sestavování jednotlivých bloků instrukcí tak paravirtualizovaný systém zohledňuje skutečnost, že neběží přímo na fyzickém hardware a tomu přizpůsobí běh vlastních procesů.

V případech, kdy je potřeba interpretovat danou instrukci hypervizorem, je použito speciálně vytvořené volání – takzvaný hypercall. Jedná se o zvláštní druh systémového volání, které však předává řízení přímo hypervizoru. To přináší oproti standardnímu vnitřnímu přerušení výhody, především co se týče efektivity a rychlosti zpracování. Tento přínos je zajištěn především tím, že je značně snížena režie na přepínání mezi privilegovaným a uživatelským módem. Další značnou výhodou, kterou paravirtualizace přináší, je možnost provedení několika privilegovaných instrukcí naráz. U úplné virtualizace by bylo nutné pro každou privilegovanou instrukci vyvolat individuální přerušení. V praxi je hypercall realizován, obdobně jako systémové volání, pomocí

hypercall page, která pomáhá hypervizoru identifikovat, z jakého důvodu je hostovaným systémem volán.

Díky vzniku dodatečného volání hypervizoru (hypercall), je u metody paravirtualizace také mnohem jednodušší volání jádra paravirtualizovaného systému. To je prováděno nativně pomocí obvyklého systémového volání. U metody úplné virtualizace, kde bylo systémové volání nahrazeno voláním hypervizoru, docházelo v případě volání jádra systému k vícenásobnému předávání řízení. Nejprve aplikace předala řízení hypervizoru a ten následně zavolal jádro hostovaného systému. V případě paravirtualizace je tak hypervizor volán až ve chvíli, kdy jádro hostovaného operačního systému, kterému byla předána kontrola, potřebuje provést některou z kritických instrukcí.

Nemalou výhodou je též existence takzvaného virtuálního času v prostředí paravirtualizace. Vzhledem k tomu, že paravirtualizovaný systém ví o svém běhu ve virtuálním prostředí, může tomuto stavu uzpůsobit i plánování svých procesů a využití zdrojů. Ve virtualizovaném prostředí bývá na jednom fyzickém zařízení provozováno obvykle více logických systémů. Ty se musí dělit o své zdroje a každému z nich tedy náleží pouze určitý díl těchto zdrojů. V případě úplné virtualizace však tyto hostované systémy nemají o existenci dalších hostovaných systémů informace. Důsledek této skutečnosti je názorně vidět například na výpočtu odhadovaného času do konce nějaké operace, kdy tento výpočet hostovaný systém provádí při zohlednění veškerých dostupných zdrojů. Oproti tomu paravirtualizovaný systém si udržuje svůj virtuální čas, z kterého veškeré tyto informace odvozuje. Tento čas vždy při možnosti využití zdrojů zaktualizuje se skutečným systémovým časem, který udržuje hypervizor.

Dalším přínosem paravirtualizace je samotné zavádění hostovaného stroje. Vzhledem k historickému vývoji architektury x86, je při zavádění operačního systému procesor spouštěn v režimu reálných adres (reálný režim). Ten je až ve chvíli předání řízení operačnímu systému přepnut do chráněného režimu. To přináší ve virtualizovaném prostředí určité překážky, jelikož virtualizér musí být schopen tento reálný režim emulovat. To se však týká pouze úplné virtualizace, jelikož v případě paravirtualizace můžeme díky modifikacím paravirtualizovaného systému tento spouštět rovnou v chráněném režimu. Stejně tak lze u paravirtualizace postrádat BIOS, jelikož veškeré informace o hardware se dají předat pomocí jiných metod.

### 4.3.2 Paravirtualizace paměti

Výraznou změnu, oproti dříve popsaným metodám virtualizace, přináší paravirtualizace v oblasti práce s paměťovými zdroji. Hlavním přínosem je technologie ballon driver, která umožňuje dynamicky měnit velikost dostupných paměťových prostředků za běhu systému. Technologie ballon driver pracuje na principu definování dvou hodnot při návrhu virtualizovaného prostředí. První hodnota určuje minimální počáteční velikost dostupné paměti, druhá hodnota pak říká, jaká je nejvýše přidělitelná velikost paměti. Rozdíl mezi těmito hodnotami představuje paměťový prostor, který může, ale také nemusí, být virtuálním strojem využit. V případě, že tento paměťový prostor není obsazen, umožňuje technologie ballon driver jeho využití hypervizorem nebo prostřednictvím hypervizoru jinou virtualizovanou entitou. V případě, že hostovanému systému dochází volné paměťové prostředky, požádá hypervizor o přidělení další volné paměti. Hypervizor zašle instrukci ballon driveru a ten příslušnou paměť uvolní a poskytne ji systému. Tato metoda může přinášet mnoho výhod především v případech, kdy fyzické zařízení hostuje několik paravirtualizovaných jednotek, jejichž výpočetní špičky se v čase liší. Pro každou jednotku je pak vyhrazen individuální minimální paměťový prostor, kterým tato jednotka disponuje, ale který může být v případě potřeby dynamicky navýšen do prostoru alokovaného ballon driverem. Celkové nároky na dostupnou fyzickou paměť se tak oproti metodám úplné virtualizace sníží, jelikož určitá část této paměti je v čase distribuována různým jednotkám.

Díky technologii paravirtualizace je také, především díky možnosti modifikovat některé procesy v hostovaném operačním systému, mnohem jednodušší udržení konzistence paměťového prostoru. Není již potřeba používat metody stínování virtuální paměti, ale jednoduše se paravirtualizovanému systému dovolí z paměti pouze číst. V případě, že je nutné zapsat do paměti nová data, je jádrem hostovaného systému vyvolán hypercall, kterým je o zápis požádán hypervizor. Tato technologie by však sama o sobě vedla ke značnému zvýšení počtu volání hypervizoru. Proto je tento proces vylepšen o zpožděný zápis, který dovolí provádění zápisu provádět dávkově ve větším množství. Jednotlivé způsoby paravirtualizace se pak liší především v konkrétní implementaci tohoto zpracování a v zajištění včasného zpracování dávky tak, aby nedocházelo k zastarávání informací v paměti uložených.

Oproti úplné virtualizaci je také mnohem efektivnější práce s Translation Look-aside Buffer (TLB). Tato cache paměť zrychluje stránkování paměti a usnadňuje překlad virtuálních adres (v tomto případě se pojem virtuální netýká virtualizovaného systému) na fyzické tím, že ve svém prostoru udržuje seznam posledních použitých záznamů z tabulky stránek paměti. Díky tomu je přístup na posledně použitá místa v paměti mnohem rychlejší. Metoda úplné virtualizace vyžaduje při přepnutí kontextu mezi hypervizorem a jednotlivými virtualizovanými systémy vždy vyprázdnění TLB. Paravirtualizace používá způsob, který je podobný sdílení paměti aplikacemi v multitaskingových operačních systémech. Jednoduše v rámci každého hostovaného systému vyhradí jistou oblast adresního prostoru hypervizoru. V rámci přepínání kontextu pak není potřeba přepínat adresní prostor, jelikož tento se stále týká hypervizoru. Tomu pak akorát zbývá zabezpečení těchto oblastí před nežádoucím přístupem ostatních běžících procesů.

### 4.3.3 Paravirtualizace vstupně-výstupních zařízení

Paravirtualizace vstupně-výstupních zařízení je velmi populární i v případě plné virtualizace. Modifikace nebo přímé vytvoření příslušného ovladače paravirtualizovaného vstupně-výstupního zařízení je totiž velmi snadná záležitost i v případě proprietálních systémů.

Efektivita provozu paravirtualizované periferie je ve svém důsledku mnohem vyšší, než jakýkoliv způsob emulace. V případě emulace komunikace s periferií musí docházet k několika úrovním překladu instrukcí, pomocí kterých spolu systém a zařízení komunikují. V první fázi je nutné přeložit a zpracovat instrukce, které vysílá hostovaný systém směrem k domnělému (virtualizovanému) vstupně-výstupnímu zařízení. Tato volání musí hypervizor přeložit, zpracovat a především zkontrolovat, zda nenarušují některou z nutných podmínek virtualizace. Následně musí zaslat instrukce v příslušné instrukční sadě fyzickému zařízení. V případě odezvy se celý proces opakuje s tím rozdílem, že jde v opačném směru. Tento proces tak zatěžuje činnost hypervizoru v mnohem větší míře, než je nutné. Paravirtualizované ovladače využívají vědomého porušení podmínky izolace (ekvivalence) hostovaným systémem a umožňují tak komunikaci přímo s hypervizorem. Z toho důvodu si mohou dovolit komunikovat instrukcemi v abstraktní podobě, které jsou oproti instrukční sadě fyzických zařízení velmi

zjednodušená. Spoustě potenciálních problémů také mohou díky vědomí o běhu ve virtualizovaném prostředí předcházet. To se týká například přístupů do paměti, kdy nemusí přímo definovat oblast, do které se mají požadovaná data načíst, ale toto rozhodnutí nechají na hypervizoru, od kterého si později toto umístění nechají sdělit.

Stejně tak je v případě paravirtualizace usnadněna komunikace hypervizoru s fyzickými periferiemi. V případě paravirtualizace se využívá jednoho privilegovaného operačního systému, který je označován jako Dom-0 (Domain-0 - v konkrétních implementacích může mít i jiné označení). Ten je zodpovědný za správu přístupu k periferiím a hypervizor tak nemusí mít implementovány příslušné ovladače a znalosti o komunikaci s periferiemi. Nezřídka je též privilegovaný hostovaný systém (Dom-0) zodpovědný též za administraci ostatních hostovaných zařízení – řídí provoz, zapíná, vypíná, vytváří a maže ostatní hostované systémy. Pouze pro úplnost je nutné uvést, že ostatní, to znamená neprivilegované, hostované celky se označují jako Dom-U (Domain-U).

Problémem paravirtualizace však zatím zůstává určitá nejednotnost paravirtualizačních platforem různých hypervizorů. I přesto, že v posledních letech došlo v této oblasti k výraznému posunu a vzniku několika poměrně univerzálních řešení, stále zde existují určité mezery. Nejvíce se o tvorbu jednotných virtualizačních rozhraní zasadil Rusty Russell, který navrhl velmi jednoduchou abstraktní strukturu Virtio, vycházející z hypervizoru Lguest. Vytvoření jednotného komunikačního rozhraní mezi periferiemi a hypervizorem je zásadním požadavkem pro zajištění přenositelnosti a migrovatelnosti paravirtualizovaných systémů.

## *4.4 Hardwarová virtualizace*

Všechna výše popsaná softwarová řešení virtualizace architektury x86 jsou sice funkční, ale díky výše popsaným skutečnostem stále poměrně neefektivní ve srovnání s platformami, které podporují virtualizaci nativně. Již v úvodu práce byly uvedeny (v kapitole o Vývoji vizualizačních technik – 3.3) zmínky o prvních virtualizačních strojích. Historicky první plně hardwarově virtualizovatelnou platformou však byl až počítač IBM/370 uvedený na trh v roce 1970. Tento mainframový počítač řešil mnoho uvedených problémů hardwarovou cestou a jednoduchý integrovaný hypervizor (PR/SM –

Processor Resource/Systém manager) byl pouze vrstvou sledující a řešící případné konflikty. Příkladem stability a propracovanosti návrhu tohoto mainframe počítače je například možnost rekurzivní virtualizace, kdy hypervizor PR/SM je schopen vytvořit až 60 izolovaných celků a v rámci každého z nich je možné vytvořit pomocí hypervizoru z/VM další stovky virtuálních strojů. O spolehlivosti tohoto počítače svědčí i to, že se bez větších problémů používal až do 90. let 20. století, kdy byl postupně nahrazován architekturou ESA/390 a později 64bitovou architekturou zSeries.

Skutečnost, že splnění Popek-Goldbergových podmínek virtualizace hardwarovou cestou přináší nesporné výhody, si velmi záhy po uvedení prvního virtualizačního softwaru určeného pro architekturu x86 uvědomili i výrobci tohoto hardwaru. Vzhledem k obtížnosti změn v již existující 32 bitové architektuře však byla hardwarová podpora začleněna až do architektury x86_64. Jako první uvedla své řešení hardwarové virtualizace společnost Intel v listopadu roku 2005. Jednalo se o technologii Intel VT-x, s kódovým označením „Vanderpool“. Krátce po ní přispěchala s obdobnou technologií i konkurenční společnost AMD, která představila své virtualizační řešení s názvem AMD-V (respektive AMD SVM) a kódovým označením „Pacifica“ v květnu roku 2006.

Způsoby, kterými je hardwarová virtualizace zajišťována u architektury x86_64 představují následující kapitoly.

### 4.4.1 Hardwarová virtualizace procesoru

Obě virtualizační řešení, tedy Intel VT-x i AMD-V, představují v rámci hardwarové podpory virtualizace procesoru především modifikace chráněných režimů procesoru. Obě technologie zavádějí novou úroveň oprávnění (Ring -1), která je vyhrazena pro běh hypervizoru a má nejvyšší úroveň ochrany a oprávnění. Díky tomu je možné provozovat hostované virtuální systémy ve stejných úrovních ochrany, jako kdyby běžely nativně, a není tak nutné používat technologie ring deprivilegingu. Aby bylo možné tuto ochranu hypervizoru realizovat, musely být zavedeny do instrukční sady nové instrukce. Jedná se především o instrukce vmrun (nebo vmentry) a vmexit. Tyto zajišťují změnu kontextu na virtuální stroj (vmrun), respektive spouštění privilegovaného módu hypervizoru (vmexit).

Nedílnou součástí těchto instrukcí, která podporuje jejich provoz, je pak nová paměťová struktura VMCS (virtual machine control structure) některými autory (Adams, Agesen, 2006 [10]) označovaná jako VMCB (virtual machine control block), která umožňuje ukládání a opětovné načtení stavu registrů hostovaného a hostujícího systému. Tato struktura obsahuje několik registrů, které velmi usnadňují přepínání kontextů mezi hyperprivilegovaným módem a virtualizovanými systémy. Základní součásti struktury VMCS jsou následující

- **Stav virtuálního stroje** – obsahuje informace o stavu virtuálního procesoru a o přerušeních virtuálního procesoru
- **Stav hypervizoru**
- **Příznaky řízení virtuálního stroje** – obsahuje příznaky určující události, při kterých přejde řízení do root operation režimu (vmexit). Těmito událostmi může být softwarové nebo hardwarové přerušení, nemaskovatelné přerušení nebo vykonání některé z instrukcí instrukční sady
- **I/O bitová mapa** – adresa v paměti obsahující bitovou mapu I/O portů, při jejichž použití dojde k zavolání instrukce vmexit
- **Offset čítače time-stamp** – obsahuje hodnotu čítače time-stamp pro virtualizovaný stroj
- **Masky CR0 a CR4** – obsahuje masky řídících registrů, které může virtualizovaný stroj nastavovat přímo (bez volání vmexit)
- **Stínové hodnoty CR0 a CR4 -**
- **Povolené hodnoty CR3** – obsahuje hodnoty registru CR3, které může virtualizovaný stroj nastavovat přímo (bez volání vmexit)

Struktura VMCB obsahuje také speciální registr **Exit Info**, který obsahuje informace o podmínkách, které způsobí volání instrukce vmexit. Příkladem takové podmínky může být například vnější přerušení. V případě, že k nějaké z těchto podmínek dojde, informace o tom se též zapíše do struktury VMCS aby mohl hypervizor v budoucnu lépe identifikovat důvod změny kontextu.

### 4.4.2 Hardwarová virtualizace paměti

Technologie hardwarové virtualizace, vytvořené společnostmi AMD a Intel, neimplementovaly podporu virtualizace MMU (Memory Management Unit) hned v době uvedení jejich virtualizačních řešení. Tuto podporu implementovaly až v letech 2007 (AMD), respektive 2009 (Intel). Do té doby bylo používáno vlastnosti hypervizoru, kdy byla při každém přístupu k tabulce stránek vyvolána instrukce vmexit. Toto řešení však nebylo úplně ideální, jelikož jeho efektivita byla srovnatelná se softwarově prováděnou virtualizací paměti. Nově zavedené technologie dostaly pojmenování Nested Paging (v průběhu dalšího vývoje přejmenovaná na Rapid Virtualization Indexing) v případě AMD a Extended Page Tables v případě Intelu. Obě techniky jsou však svým přístupem velmi podobné a liší se jen málo.

Obě technologie rozšiřují komponentu Memory Management Unit (MMU) o možnost zpracovávat v průběhu překladu dvě tabulky stránek. První tabulka stránek, která náleží virtualizovanému systému, umožňuje překlad z virtuální na reálnou adresu (reálná adresa označuje adresu, kterou virtualizovaný stroj považuje za adresu fyzickou). Druhá tabulka stránek, Nested Page Table nebo Extended Page Table, zajišťuje překlad z reálných na fyzické adresy. V případě hardwarové virtualizace paměťových komponent je také mnohem efektivnější používání Translation Look-aside Buffer (TLB), kdy TLB obsahuje celý překlad adresy hostovaného systému na fyzickou adresu.

Oproti metodám překladu, které byly zajišťovány hypervizorem, (především stínování tabulek) jsou hardwarové metody virtualizace paměti mnohem méně náročné na činnosti hypervizoru. Ten nemusí zajišťovat pravidelné synchronizace stránek paměti, jelikož tuto činnost zajišťuje přímo MMU. Na druhou stranu však tyto technologie přidávají zvýšenou režii v přístupu k paměti. To je způsobeno potřebou zajišťovat překlad pro dvě sady tabulek. Tuto nevýhodu lze z části potlačit používáním větších velikostí stránek, čímž lze zvýšit i efektivitu TLB, jelikož při stejném množství záznamů v TLB bude tento odkazovat na větší paměťovou oblast. Mezi další techniky umožňující efektivnější zpracování dat v paměti pak patří sdílení TLB hostovanými stroji. To je umožněno pomocí rozšíření tagged TLB, které umožňuje přiřadit každému záznamu v TLB zvláštní identifikátor odkazující na příslušný hostovaný stroj. Díky tomu není nutné

při každé změně kontextu měnit obsah TLB. Každý hostovaný systém má přístup pouze k datům, která jsou v TLB označeny jeho identifikátorem. Nemůže tak dojít ke konfliktům a je tedy umožněno, aby data byla uložena v TLB i při změně kontextu.

### 4.4.3 Hardwarová virtualizace vstupně-výstupních zařízení

Pro umožnění hardwarové virtualizace vstupně-výstupních zařízení je nutné zajistit dva druhy podpory. První spočívá v podpoře chipsetu, který musí umět nějakým způsobem přiřadit patřičný hardware jednotlivým virtualizovaným strojům a zajistit překlad DMA do paměti hostovaného stroje a zpět. Za druhé musí být zajištěna podpora ze strany příslušných vstupně-výstupních zařízení, která musí umožnit provázání jednoho fyzického zařízení s více hostovanými systémy, přičemž musí být toto zařízení schopeno od sebe efektivně rozeznat jednotlivé hostované celky. V případě zajištění těchto podmínek přináší hardwarová virtualizace vstupně-výstupních zařízení výkon, který se velmi přibližuje výkonu konvenčních řešení a umožňuje též bezproblémovou identifikaci fyzických zařízení pomocí běžných ovladačů. Jedinou nevýhodu lze spatřovat ve skutečnosti, že hardwarově virtualizovaná zařízení je možné jen s velkými obtížemi migrovat. To je způsobeno metodou uložení informací o stavu komunikace s jednotlivými systémy. Tato komunikace je uložena na mnoha místech v paměti a před případnou migrací je potřeba je z paměti vyfiltrovat. Taktéž je například oproti emulaci zařízení nutné používat v rámci migrace stejný typ fyzického zařízení na všech strojích.

Podpora chipsetu pro hardwarovou virtualizaci se vyvinula z technologií, které se souhrnně označují jako IOMMU (input/output memory management unit). Tyto technologie usnadňují překlad z adresního prostoru vstupně-výstupních zařízení do fyzického adresního prostoru a naopak. Obdobný princip využívá klasická MMU, která však do/z fyzického adresového prostoru překládá adresy virtuální. Technologie IOMMU doznaly největšího rozvoje v souvislosti s grafickými kartami. Drobnou modifikací těchto technologií pak společnosti Intel a AMD vytvořily technologie, které umožňují překlad vstupně-výstupních adres do adresního prostoru jednotlivých virtuálních strojů. Společnost Intel svou technologii pojmenovala Intel VT-d (Intel Virtualization Technology for Directed I/O), společnost AMD se nejprve spokojila s obecným pojmenováním AMD IOMMU (AMD I/O Memory Management Unit), kterou v druhé generaci přejmenovala na

AMD-Vi. Druhá úroveň hardwarové virtualizace vstupně-výstupních zařízení (podpora ze strany příslušných periferií) bývá v dnešní době nejčastěji zajištěna standardizovaným řešením konsorcia PCI-SIG. Toto sdružení je zodpovědné za vývoj a definování standardů rozhraní PCI Express. Pro potřeby virtualizace definovalo toto konsorcium tři standardy, které publikovalo pod označením IOV (I/O Virtualization).

Základním problémem, který IOMMU řeší, je proces DMA přenosu. V případě přímého přístupu do paměti ze strany zařízení je pro toto zařízení nutná znalost počáteční adresy v paměti, odkud (respektive kam) bude přenos směřovat. V případě konvenčního (nevirtualizovaného) řešení je zařízení předána fyzická adresa v paměti. Problémem virtualizace však je, že virtualizovaný systém tyto adresy nezná a přistupuje k nim prostřednictvím hypervizoru (metody překladu v případě jednotlivých virtualizačních metod byly popsány v předcházejících kapitolách). Výše popsané možnosti (emulace, paravirtualizace) jsou v přístupu k překladu DMA požadavků velmi neefektivní. Oproti tomu jednotka IOMMU tento překlad řeší poměrně efektivně, a hlavně bez využití dodatečných softwarových a hardwarových zdrojů. Tato jednotka pracuje na principu doménového rozdělení jednotlivých hostovaných strojů, kdy ke každému logickému celku, který je na daném stroji hostován, je přiřazen vlastní doménový prostor s vyhrazenými zdroji. Pomocí domén je pak velmi jednoduché identifikovat, který hostovaný stroj předává nějaký požadavek. Vzhledem k tomu, že hypervizor vytváří každému hostovanému systému vlastní tabulku stránek, stačí pouze tyto tabulky předat IOMMU a tato pak realizuje samotný překlad. Ten je realizován analogicky jako v případě klasické MMU, s využitím většiny efektivních technik, jako je například využití cache paměti (IOTLB), která obsahuje naposledy provedené překlady.

Jak již bylo nastíněno, efektivita hardwarové virtualizace vstupně-výstupních zařízení nespočívá pouze na straně chipsetu, ale také na straně těchto zařízení. Mezi nejpoužívanější zařízení se v posledních letech zařadily, především díky své univerzálnosti, zařízení s komunikačním rozhraním PCI Express. Konsorcium PCI-SIG, které je za vývoj těchto zařízení zodpovědné, vypracovalo specifikaci pro hardwarovou virtualizaci těchto zařízení IOV, která se skládá ze 3 částí.

### Address Translation Services (ATS)

Tato technologie překladu vychází z výše popsaného principu překladu adres pomocí IOMMU, konkrétně implementace IOTLB. ATS využívá IOTLB cache paměť, která je integrována přímo do vstupně-výstupního zařízení (Device-IOTLB). Zařízení tak může IOMMU požádat o překlad určitého rozsahu paměti a ten si uložit do své IOTLB paměti. V případě DMA přístupu do paměti tak zařízení přistupuje do správného adresního prostoru bez nutnosti překladu. Aby byla zohledněna bezpečnost a nedošlo k případným konfliktům, musí při zahájení přenosu zařízení informovat IOMMU o skutečnosti, že DMA požadavek obsahuje již přeložený paměťový prostor.

### Single Root IOV (SR-IOV)

Tato technika je založena na partitioningu (viz kapitola 5.2) vstupně-výstupního zařízení. Toto PCI Express zařízení je pomocí I/O řadiče rozděleno na několik virtuálních logických entit, které jsou poté přiřazeny jednotlivým hostovaným strojům. Ty pak k zařízení přistupují stejně, jako by ho měli ve výhradním užívání a díky partitioningu nemůže dojít ke konfliktu s dalšími systémy. Celá tato operace je umožněna díky tomu, že dané vstupně-výstupní zařízení umožňuje virtualizaci (partitioning) svého operačního prostoru. Díky tomu dokáže vytvořit omezené množství entit, které se tváří jako fyzické zařízení a obsahují vlastní identifikátory a adresní prostor. Celá tato technika převyšuje z pohledu efektivnosti všechny dosud popsané metody virtualizace vstupně-výstupních zařízení. Důvodem je především to, že mimo úvodní inicializace zařízení jednotlivým hostovaným strojům nevyžaduje tato technika komunikaci s hypervizorem.

### Multi Root IOV (MR-IOV)

Principielně stejná technologie jako Single Root IOV. Jediný rozdíl spočívá ve skutečnosti, že zatímco SR-IOV umožňuje komunikaci v rámci jedno fyzického stroje, Multi Root IOV technologie je přímo navržená pro cloudové systémy se sdílenou PCI Express sběrnicí. Umožňuje tak efektivní využití zařízení více fyzickými stroji.

## 4.5 Partitioning

Stejně jako technologie emulace, není ani partitioning typicky virtualizační metodou. Jeho zařazení do této práce je však vzhledem k souvislostem poměrně na místě. Stejně jako u emulace však není třeba se touto technikou zabývat blíže, nicméně její zmínění je vzhledem k souvislostem přinejmenším vhodné.

Metody takzvaného soft-partitioningu se v prostředí architektury x86 vyskytovaly již dlouho před uvedením prvního reálně virtualizačního řešení pro tuto platformu. Praktické užití tato metoda přináší v případech, kdy není potřeba virtualizovat celé operační systémy, ale z hlediska jejich běhu a nezávislosti je vhodné jejich faktické oddělení. Takové systémy tak používají stejné jádro operačního systému, ale jinak jsou na sobě nezávislé a běží odděleně. K izolaci jednotlivých logických celků se využívá vytvoření kontextů, které jsou na zbývajících kontextech nezávislé a v případě nastavení o nich prakticky nevědí. Každému kontextu jsou přiřazeny zdroje (adresový prostor, souborový systém, proces, vlákno, …) a vytvořen takzvaný identifikátor kontextu, kterým se identifikuje vůči částem systému společných pro všechny kontexty. Prostředky, kterými je prostředí kontextu ohraničeno (v OS Linux), jsou následující:

**Izolace procesů**

S výjimkou procesu *init*, který je z důvodu kompatibility viditelný ve všech kontextech, jsou veškeré procesy běžící v jednotlivých kontextech izolované. V praxi je to nejlépe patrné z výpisu procesů (/proc), kde jsou viditelné pouze procesy běžící v daném kontextu. Stejně tak nelze procesům z jiných kontextů zasílat signály a zprávy, či je případně krokovat.

**Izolace zdrojů**

Mezi další zdroje, které je nutné izolovat, patří například sdílená paměť, zprávy, semafory (souhrnně SYSVIPC), virtuální terminály, sockety a další.

**Ireversibilní chroot**

Z hlediska izolace adresářové struktury jednotlivých kontextů se používá standardních metod *chroot* (change root), které umožňují přiřadit procesům jako kořenový adresář jiný adresář v rámci sdíleného souborového systému. Adresář, který je ve společné adresářové struktuře nadřazený, je pak chráněn mechanismem, který zabraňuje opuštění prostředí změněného kořenového adresáře. Oproti klasickým virtualizačním technikám se však jedná z hlediska bezpečnosti o určité ústupky, jelikož existují techniky, které umožňují prolomení této bariéry.

**Omezení sítě**

Z hlediska oddělení kontextů je podstatné izolovat síťový provoz. Toho je v praxi docíleno vytvořením aliasů, které umožňují přiřadit jednotlivým kontextům různé IP adresy. Jednotlivé kontexty tak nemohou zasahovat do síťového provozu okolních kontextů. Jediným problémem tvorby aliasů je localhost (127.0.0.1), který musí být z principu společný pro všechny kontexty. Tento problém se však dá poměrně snadno vyřešit různými administrativními opatřeními.

**Capability ceiling**

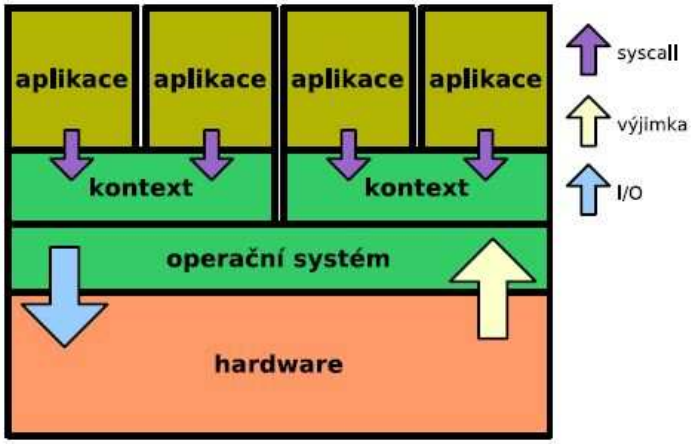
Tato technika umožňuje omezit jednotlivé kontexty ze strany přístupu k jádru. Bylo by jistě nepříjemné, kdyby kdokoliv z kteréhokoliv kontextu mohl například zablokovat přístup k určitým blokovým zařízením nebo dle libosti vypínat celý fyzický stroj. Primárně tak tato technika umožňuje měnit oprávnění uživatele root jednotlivých logických kontextů a omezovat je v používání rebootu, vytváření přístupových bodů pro zařízení (device nod), práce se souborovými systémy, nastavování síťových parametrů a úpravy jádra.

**Globální limity**

Slouží k omezení prostředků, které jsou jednotlivým kontextům vyhrazeny. Jedná se například o velikost paměti využitelné jedním kontextem, maximální využití procesorového času, maximálního počtu otevřených souborů a další.

Prakticky je existence kontextů realizována testováním, které při každém pokusu o přístup k některému ze zdrojů (paměť, procesor, vstupně-výstupní zařízení) testuje

příslušnost daného procesu k určitému kontextu. Oproti virtualizačním technikám je implementace tohoto řešení podstatně jednodušší. To je vidět především v souvislosti s paravirtualizací, která také využívá metody modifikace hostovaného systému. Zavedení algoritmu, který testuje příslušnost procesu k danému kontextu, nebývá složité a mnoho jader operačních systému takový algoritmus obsahuje nativně.



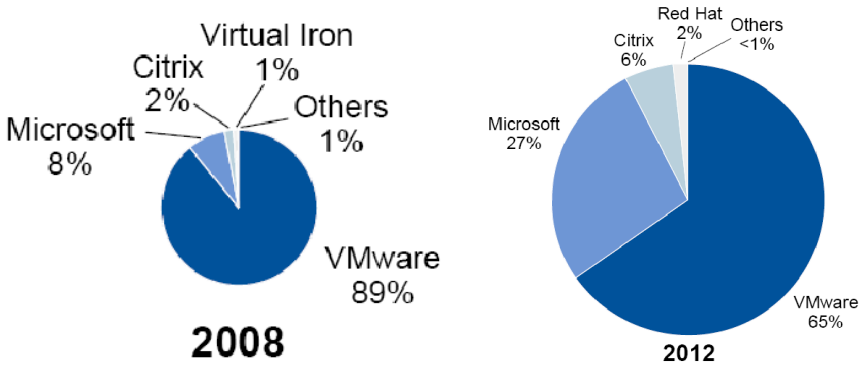

Obrázek č. 6 – Schéma partitioningu – zdroj Děcký [8]

Výhody, které toto řešení přináší, jsou nasnadě. Pomocí této metody se výrazně zvyšuje bezpečnost jednotlivých kontextů, jelikož v případě ohrožení jednoho z kontextů nelze infikovaný či nestabilní proces přenést do kontextů okolních. Rovněž realizace ochrany částí souborového systému je mnohem bezpečnější oproti standardní ochraně pomocí přístupových práv. Podstatnou výhodu též přináší fakt, že lze do samotného kontextu vyčlenit jen určité aplikace. Tím vzniká podstatná úspora zdrojů, jelikož se „virtualizují“ jen určité procesy a není potřeba znovu virtualizovat celé jádro operačního systému. Nejčastěji se takto virtualizovaná prostředí využívají v případě web hostingových služeb, kdy na jednom fyzickém serveru běží větší množství oddělených kontextů, které poskytují přístup k jednotlivým hostovaným stránkám, ale navzájem se nemohou ovlivnit.

# 5 Analýza trhu virtualizačních řešení

Trh virtualizačních produktů je trhem, který se v posledních letech překotně vyvíjí a na kterém se střetává několik zásadních přístupů, které jsou vlastní celému trhu s informačními technologiemi. Definován je tento trh organizacemi, *„které hledají řešení umožňující virtualizaci jejich aplikací s využitím existujících x86 serverů a operačních*

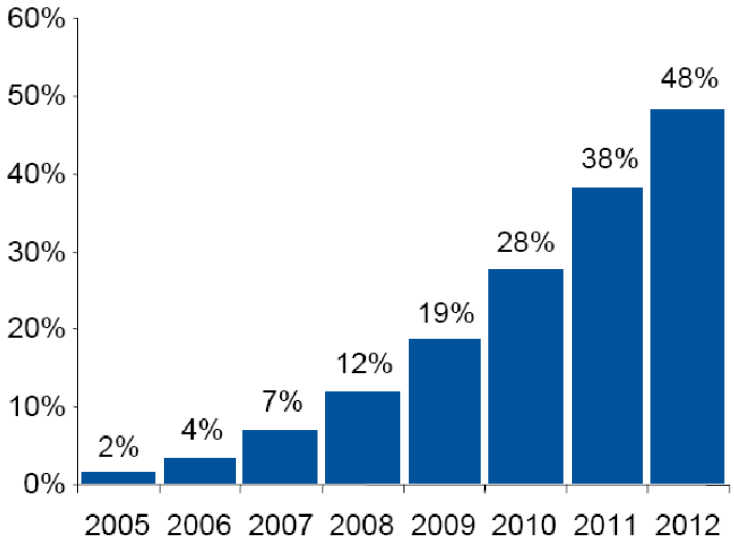
*systémů, při menším množství hardwaru, lepším využitím zdrojů a větší flexibilitě*“ (Bittman, Dawson, Weiss, 2010, str. 2 [11]). Na trhu virtualizačních řešení existuje společnost, která je díky svému technologickému náskoku v pozici leadera a která má majoritní podíl na trhu (VMware). Dále zde existuje společnost, která na trh vstoupila mnohem později, ale která má velmi širokou síť produktů a zákazníků, což představuje výhodu z obchodního hlediska (Microsoft). Na trhu působí také společnosti, jejichž virtualizační řešení je v samotném základu open-source, ale tyto společnosti mu přidávají nějakou důležitou přidanou hodnotu (Citrix, Red Hat). Stejně tak zde působí samotné komunity, které výše zmíněné open-source řešení dále rozpracovávají a nabízejí potenciálním klientům (KVM). Mimo tyto hlavní složky nelze opomenout společnosti, které nemají nijak velký podíl na trhu, ale které buď díky vhodným akvizicím, patentovým a jiným dohodám nebo díky nějaké přidané hodnotě na trhu působí a mají své poměrně stálé uživatele (Oracle, Novell, Parallels).




Obrázek č. 7 - Tržní podíly v roce 2008 a 2012 – zdroj www.gartner.com

Dá se očekávat, že tržní podíly, které jsou v současné době poměrně stabilizovány, se budou v následujících letech pozvolna měnit. Důvodem je jak zatím nedostatečná penetrace trhu, tak skladba zákazníků, kteří používají virtualizovaná řešení. Hlavní příčinou, která umožní změnu tržních podílů, však bude hlavně očekávané rozšíření trhu, u kterého se v průběhu několika málo let očekává několikanásobný nárůst. V současné době využívají virtualizačních řešení především velké společnosti, které sice jednotlivě zaujímají velké tržní podíly, ale ve svém součtu je jejich podíl na trhu virtualizace menšinový. Naopak malé a střední podniky, které v součtu tvoří hybatele trhu, jsou s nasazováním virtualizace teprve v počátcích, nebo se na nasazení teprve chystají. Toho

pravděpodobně využijí společností, které v současné době mají na trhu virtualizačních produktů menší podíly, ale jejich vztah k menším a středním podnikům je díky jejich dalším produktům mnohem silnější.




Obrázek č. 8 - Procento vizualizovaných serverů - zdroj www.gartner.com

## 5.1 Hlavní hybatelé trhu

V rámci této kapitoly budou stručně představeny společnosti, které jsou v současné době hybateli trhu s virtualizačními produkty.

### 5.1.1 VMware

Společnost VMware, jak už bylo výše několikrát zmíněno, je označována za průkopníka virtualizačních řešení na architektuře x86. Založena byla roku 1998 a v květnu 1999 bylo představeno její první virtualizační řešení VMware Workstation. V roce 2001úspěšně vstoupila do oblasti virtualizace serverů s dvěma produkty. Prvním z nich byl VMware GSX server, který vyžadoval ke svému spuštění hostitelský systém (hosted) a v posledních letech byl přejmenován na VMware server. Tento projekt je stále velmi oblíben, ale společnost VMware v letošním roce ukončí jeho podporu a doporučuje přejít na novější produktovou řadu. Druhým serverovým řešením, uvedeným v roce 2001 byl produkt VMware ESX server, který v sobě obsahoval již základní operační systém

(hostless) a hypervizor tak běžel přímo na hostitelském hardware. V roce 2004 byla společnost VMware koupena společností EMC Corporation a v roce 2007 byla část jejích akcií uvolněna na New Yorské burze.

V současné době poskytuje společnost VMware množství produktů, které jsou určeny pro různé užití. Stěžejním produktem je VMware ESX server, který je určen pro nasazení ve velkých podnikových sítích. Jeho součástí je platforma vCenter, která obsahuje velkou přidanou hodnotu v podobě management tools, usnadňující nasazení, migraci, správu zdrojů, automatizaci a další. Jako alternativa k produktu VMware ESX server byla uvolněna jeho bezplatná verze VMware ESXi server, která je určena především pro menší nasazení. Tato verze je sice zdarma, ale je ochuzena o mnoho užitečných nástrojů z management tools. Již byl také zmíněn produkt VMware Server (dříve GSX), kterému však v tomto roce končí podpora a jeho nasazení na novějších systémech je již nyní trochu problematičtější. V rámci serverových produktů je potřeba zmínit také produkt vSphere, který je založen na ESX serveru a je určen pro vytváření cloud-computing řešení. Mimo tyto produkty nabízí společnost VMware také produkty pro desktopovou virtualizaci (VMware Workstation, VMware Fusion, VMware Player) a aplikační virtualizaci (ThinApp). Všechna tato fakta tuto společnost předurčují k jasné roli leadera na trhu virtualizačních řešení.

### 5.1.2 Microsoft

Společnost Microsoft netřeba představovat. Jedná se o společnost, která zaujímá větší či menší tržní podíl snad v každém odvětví informačních technologií. Na trh virtualizačních produktů však vstoupila poměrně pozdě a tuto ztrátu oproti konkurenci dohání i v současné době. Jejím prvním virtualizačním produktem byl produkt Microsoft VirtualPC, který byl původně vyvíjen společností Connectix a v roce 2003 zakoupen společností Microsoft. Tento produkt, který byl původně určen především pro desktopovou virtualizaci, však k výraznějšímu zasáhnutí trhu, i přes svou značnou oblibu mezi uživateli, nestačil. V roce 2005 tak společnost Microsoft vydala Microsoft Virtual Server 2005, který byl určen především pro podnikové nasazení a vycházel z původního VirtualPC. Jako reakce na kroky konkurentů na trhu virtualizačních řešení byl Microsoft Virtual Server 2005 ve verzi R2 v roce 2006 uvolněn k bezplatnému použití.

V roce 2008 pak společnost Microsoft představila své další virtualizační řešení Microsoft Hyper-V Server, kterým do značné míry dohnala náskok konkurence v oblasti hardwarové virtualizace. Tento hypervizor byl původně součástí operačního systému Microsoft Windows Server 2008, ale později byl uvolněn i jako samostatně fungující produkt pod názvem Microsoft Hyper-V Server 2008. Tato samostatně fungující varianta byla uvolněna k bezplatnému použití, ale oproti verzi integrované v Microsoft Server 2008 má v mnohém omezenou funkcionalitu.

### 5.1.3 Xen

Virtualizační produkt Xen byl a je vyvíjen jako open source projekt. Celý projekt byl založen v roce 2002 na University of Cambridge. Později byl vývoj projektu řízen společností XenSource, která vznikla za tímto účelem. V roce 2003 bylo uvolněno první veřejné vydání hypervizoru Xen, který byl však kvůli různým omezením vhodný především pro desktopovou virtualizaci. V roce 2005 byl komunitou vydán produkt Xen v3 určený pro enterprise nasazení. Ten již podporoval hardwarovou virtualizaci, a parametry hostovaných počítačů byly na svou dobu poměrně solidní. Problematická však byl virtualizace jiných než linuxových strojů. To se změnilo v roce 2006, kdy společnost XenSource uzavřela partnerství se společností Microsoft, na základě kterého byl vydán produkt XenEnterprise 3, který podporoval i virtualizaci produktů společnosti Microsoft. K tomu byla přidána rozšířená podpora management tools, například v podobě produktů XenOptimizer a XenMotion.

V roce 2007 byla společnost XenSource koupena společností Citrix, která převzala hlavní zodpovědnost za vývoj hypervizoru Xen. Stávající produkty byly přejmenovány a již pod vedením společnosti Citrix byly vydány další verze produktu XenServer (5.0, 5.5). Na vývoji hypervizoru Xen však participují i další společnosti, jako jsou IBM, Intel, HP, Oracle, Novell, Red Hat a další. Tyto společnosti se většinou soustředí na vývoj management tools a přidané hodnoty produktu. Samotný vývoj hypervizoru tak leží především na komunitě. I díky tomu je samotný hypervizor Xen bezplatný.

### 5.1.4 Časový přehled

Jen pro doplnění je zde uveden časový přehled hlavních milníků a akvizicí v průběhu posledních let.

| čas | VMware, ESX | Citrix, Xen | Microsoft, Hyper-V |
|---|---|---|---|
| 1998 | založení společnosti Vmware | | |
| 1999 | Workstation 1.0 | | |
| 2001 | GSX Server 1.0<br>ESX Server 1.0<br>Workstation 3.0 | | VirtualPC (Connectix) |
| 2003 | ESX Server 2.0<br>VirtualCenter a vMotion<br>Workstation 4.0 | open source Xen | Microsoft kupuje VM od Connectix |
| 2004 | EMC kupuje VMware<br>Workstation 4.5 | | VirtualPC 2004 |
| 2005 | ESX 2.5<br>GSX 3.2<br>Workstation 5.5 | | Virtual Server 2005 |
| 2006 | VMware Infrastructure 3<br>VirtualCenter 2.0<br>VMware Server 1.0 (GSX) | XenSource XenServer 3.0 | Microsoft a Xen uzavírají dohodu o spolupráci |
| 2007 | VMware Infrastructure 3.5<br>VirtualCenter 2.5<br>Workstation 6.0 | Citrix kupuje XenSource | VirtualPC 2007<br>SCVMM 2007<br>Virtual Server 2005 R2 SP1 |
| 2008 | VMware Server 2.0<br>Workstation 6.5 | Citrix XenServer 4.1 | Hyper-V 1.0<br>SCVMM 2008<br>Virtual PC 2007 SP1 |
| 2009 | vSphere 4.0<br>vCenter 4.0<br>Workstation 7 | Citrix XenServer 5.5 | Hyper-V 2.0 (R2)<br>SCVMM 2008 R2<br>Windows Virtual PC |

## 5.2 Postavení virtualizačních produktů na trhu (Magic Quadrant)

Postavení jednotlivých virtualizačních produktů na trhu poměrně přesně vystihuje studie provedená společností Gartner v roce 2010. Tato studie vychází z metody Magic Quadrant, která byla společností Gartner vytvořena speciálně pro hodnocení postavení technologických společností na trhu. Jako hodnotící body jsou použita kritéria zohledňující u každé společnosti přístup k trhu a schopnost rozhodování v první hodnotící množině a cíle a vize jednotlivých společností v množině druhé. Jednotlivé společnosti jsou poté rozřazeny do jednotlivých kvadrantů matice, které charakterizují jejich postavení. Z důvodu rapidního technologického vývoje a tím i vývoje trhu s různými technologiemi, vytváří společnost Gartner zhruba každé 2 roky aktualizovaná vydání jejich analýz.

### 5.2.1 Kvadranty matice

Jednotlivé kvadranty matice rozřazují, podle výsledků sledovaných parametrů, společnosti do 4 skupin. Jednotlivé skupiny a jejich stručná charakteristika:

**Leaders (Vůdci)**

Většinou se jedná o velké společnosti, které rozvíjejí své postavení na trhu, mají dlouhodobou vizi a potenciál růstu. V obou množinách hodnotících kritérií získávají vysoké hodnocení. Nezřídka mají dominantní postavení na trhu.

**Challengers (Vyzyvatelé)**

Jedná se většinou o velké společnosti, které mají silné postavení na trhu, ale ve zkoumaném odvětví nejsou dostatečně etablovaní. Mají silnou orientaci na trh a schopnost na trhu dosáhnout svých cílů. Chybí jim však dlouhodobá vize, která buď nebývá jasně formulována, nebo se v ní vyskytují konfliktní zájmy. Tyto společnosti mívají vysoké skóre v množině hodnotící schopnosti na trhu a nízké skóre v množině hodnotící vize a cíle podniku.

**Visionaries (Vizionáři)**

Typicky se jedná o menší podniky, které mají nápady a vize a kladou důraz na inovace. Kvůli jejich velikosti však mají problém s tržní orientací a schopnostmi

dosáhnout na trhu postavení, které by jim zajišťovalo pozici leadera. Tyto společnosti mají vysoké skóre v množině hodnotící vize a inovativní přístup a nízké v množině hodnotící tržní kritéria.

**Niche players (Dodateční hráči)**

Jedná se z valné části o podniky, které na trh vstupují. Není to ovšem pravidlem a může se jednat i o malé podniky, které z důvodu nějaké výhody drží určitou marginální pozici na trhu. Tyto společnosti jsou charakterizovány nízkým skóre v obou hodnotících množinách.

### 5.2.2 Hodnotící kritéria

**Tržní orientace**

V této množině kritérií se hodnotí především schopnosti protržního chování společnosti. Kritéria jsou následující:

**Produkty / Služby**

Schopnost podniku nabízet na trhu úspěšné zboží a služby. To zahrnuje jak současné produkty, jejich kvalitu a schopnosti, ale také například síť OEM partnerů.

**Životaschopnost**

Zahrnuje komplex vlastností, které definují podnik jak ze strany podnikové struktury (Business unit), tak z pohledu finanční strategie a organizace.

**Prodej / Ceny**

Zahrnuje schopnosti společnosti prodat své zboží, zorganizovat strukturu prodeje, uzavírat nové smlouvy a vhodně využívat různé možnosti podpory prodeje.

**Tržní reakce**

Hodnotí schopnosti společnosti reagovat na změny trhu a jeho vývoj. Hodnotí flexibilitu společnosti a schopnost využívat příležitosti, které jí trh poskytuje.

**Marketingové schopnosti**

Hodnotí kvalitu, efektivitu, kreativnost a úspěšnost marketingových akcí společnosti. Hodnotí jak schopnost prodat své produkty, tak schopnost propagace

své značky a svých produktů. Důležitá je také schopnost vzájemné pozitivní identifikace se svými zákazníky.

**Zákaznická podpora**

Zahrnuje schopnost společnosti udržovat pozitivní kontakt se zákazníky i v době po koupi produktu. Jedná se především o technickou podporu nebo zpřístupnění dokumentace. Celkově se také hodnotí případné programy podpory zákazníků, které umožňují hromadné oslovení zákazníků

**Výkonnost**

Schopnost společnosti dosáhnout stanovených cílů s efektivním využitím zdrojů a příležitostí.

Váha jednotlivých kategorií je uvedena v následující tabulce.

| Hodnotící kritéria | Váha |
|---|---|
| Produkty / Služby | vysoká |
| Životaschopnost | vysoká |
| Prodej / Ceny | vysoká |
| Tržní reakce | nízká |
| Marketingové schopnosti | vysoká |
| Zákaznická podpora | střední |
| Výkonnost | nízká |

Zdroj Gartner [11]

**Vize a cíle**

V této množině se hodnotí kritéria, která se přímo vztahují k hodnocenému odvětví a jeho produktům. Jedná se především o kritéria, která pomáhají rozvíjet produkt, dodávat mu přidanou hodnotu a schopnost prodat ho zákazníkům. Jedná se o následující kritéria:

**Znalost trhu**

Schopnost pochopit, jaké jsou potřeby a touhy zákazníků a nabídnout jim produkt, který očekávají.

**Marketingová strategie**

Vhodná a efektivní komunikace s potenciálními a stávajícími zákazníky, která zahrnuje jak klasické marketingové kanály, ale v poslední době především nová média.

**Obchodní strategie**

Zahrnuje obchodní strategie, které umožňují prodej výrobků. Hodnotí kvalitu přímých a nepřímých obchodních kanálů a schopnost zasáhnout co největší segment trhu.

**Nabídka produktů**

Hodnotí jak šíři produktů a jejich specifika, tak jejich přehlednost pro zákazníka z hlediska vhodnosti použití, využitých technologií a nabízených služeb.

**Obchodní model**

Hodnotí solidnost, výkonnost a náročnost použitého obchodního modelu.

**Vertikální strategie**

Schopnost podniku efektivně využívat své zdroje, znalosti a schopnosti k zasažení různých segmentů trhu.

**Inovace**

Hodnotí schopnost zavádět inovace, kvalitu výzkumu a vývoje, efektivitu využití finančních prostředků určených na vědu a výzkum. Nedílnou součástí je také schopnost ochránit svůj výzkum pomocí patentů a ochranných známek.

**Regionální strategie**

V této kategorii se hodnotí schopnost podniku využívat zdroje a schopnosti dostupné v jednotlivých globálních regionech. Toto zahrnuje jak schopnost výroby a prodeje v těchto regionech, tak také schopnost domluvy s regionálními partnery.

Váhy jednotlivých kritérií jsou opět uvedeny v následující tabulce

| Hodnotící kritéria | Váha |
|---|---|
| Znalost trhu | vysoká |
| Marketingová strategie | vysoká |

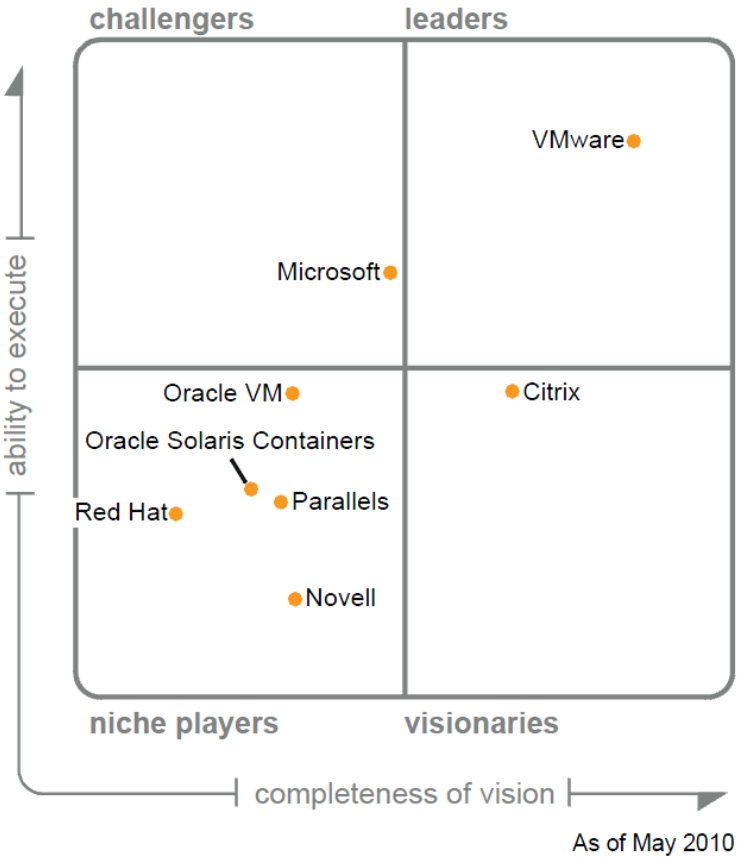
| | |
|---|---|
| Obchodní strategie | střední |
| Nabídka produktů | střední |
| Obchodní model | střední |
| Vertikální strategie | střední |
| Inovace | střední |
| Regionální strategie | nízká |

Zdroj Gartner [11]

K hodnotícím kritériím je potřeba doplnit ještě jednu informaci. Společnost Gartner ve své analýze sama upozorňuje na drobný handicap, který z nastavení analýzy vyplývá pro open source projekty jako jsou Xen nebo KVM. Vzhledem k tomu, že projekty vyvíjené komunitou aplikují úplně odlišný business model a používají úplně jiné marketingové prostředky (pokud vůbec nějaké používají), jsou v hodnocení některých kritérií předem diskvalifikované. Tento handicap je částečně vyvážen častými akvizicemi open source produktů ze strany velkých společností. Příkladem je například společnost Citrix a její vztah k hypervizoru Xen nebo společnost Red Hat a její vztah k hypervizoru KVM. I přesto je však nutné brát tuto skutečnost v úvahu při hodnocení výsledků analýzy.

### 5.2.3 Výsledky analýzy

Definujme nejprve obsah jednotlivých kvadrantů tak, jak jej analyzovala společnost Gartner.




Obrázek č. 9 - Magic quadrants – zdroj Gartner [11]

**Leaders**

Není překvapením, že jediným zástupcem v tomto segmentu je společnost VMware. Její pozice čistého leadera je zapříčiněna především tvorbou trhu virtualizačních řešení. Díky tomu, že měla tato společnost možnost trh utvářet, získala nezanedbatelný náskok jak v případě tržního podílu, ale díky tomu i v případě zkušeností s virtualizací podnikových platforem. Do budoucna musí mít tato společnost, i přes velkou konkurenci v oboru, snahu udržet si vůdčí roli. Měla by rozvíjet své portfolio zákazníků a více se soustředit na segment malých a středních podniků, kde má ve svém obchodním modelu rezervy.

**Challangers**

Mezi vyzyvateli se objevila společnost Microsoft. Toto umístění vyplývá z jejího obecného umístění na trhu informačních technologií. Díky své silné pozici v jiných

oblastech informačního průmyslu má obrovské zkušenosti v oblasti vývoje softwaru a jeho následného prodeje. Ačkoliv na trh virtualizačních produktů pronikla poměrně pozdě, podařilo se jí v průběhu několika málo let vytvořit velmi konkurenceschopný produkt (Hyper-V), kterým může oslovit širokou základnu svých zákazníků.

**Visionaries**

V oblasti vizionářů se vyskytuje hypervizor Xen. Zásluhu na tomto umístění má především akvizice společností Citrix, která tomuto produktu přidala velkou přidanou hodnotu a dokázala s ním oslovit poměrně velký segment trhu (zbývajícího po odečtení produktů společností VMware a Microsoft). Díky vývoji, který je prováděn ve spolupráci s dalšími velkými hráči na trhu informačních technologií (IBM, Intel, Novell, …), a velké komunitě vývojářů, kteří velmi dobře znají potřeby uživatelů, má do budoucna potenciál prosadit se mnohem více v komerční sféře. Problematický by pro tento typ hypervizoru mohl být pouze potenciální souboj o místo na trhu s dalším z open-source hypervizorů KVM, za kterým stojí společnost Red Hat. Vzhledem k současnému vývoji samotného linuxového jádra, do kterého již byla implementována podpora KVM, je situace poměrně zneklidňující. „*Otázka zní, jak se společnost Citrix dokáže zachovat a zda zvládne zhodnotit svou momentální výhodu tváří v tvář VMware u velkých podniků, Microsoftu u malých podniků a Red Hatu v oblasti open source.*“ (Bittman, Dawson, Weiss, 2010, str. 6 [11])

**Niche Players**

V oblasti hráčů vyplňujících mezery na trhu se vyskytuje množství různorodých společností. Jsou zde jak poměrně nové společnosti (Red Hat, Oracle), tak společnosti působící na trhu virtualizačních řešení již dlouhou dobu (Novell, Parallels). I cíle těchto společností jsou do jisté míry velmi odlišné. Zatímco například Parallels se soustředí především na jeden segment trhu (uživatelé Apple), společnost Novell se soustředí především na jeden produkt (unifikovaný management tools program). „*Prostoru na tomto rozvíjejícím trhu tak, i přes velkou konkurenci, mají tyto společnosti dost. Aby byly úspěšné, musí se však*

*navzájem dostatečně odlišit a plnit odlišné potřeby trhu.*“ (Bittman, Dawson, Weiss, 2010, str. 6 [11])

## 5.3 Srovnání předních virtualizačních produktů

V této kapitole budou stručně popsány vlastnosti nejpoužívanějších virtualizačních nástrojů. Zahrnuty jsou nástroje VMware ESX Server, Citrix XenServer a Microsoft Hyper-V Server. Do srovnání by se dalo zahrnout jistě více nástrojů. Mnoho dalších nástrojů však cílí na odlišný segment trhu a srovnání jejich vlastností by nebylo relevantní.

Jak je vidět v následujícím srovnání parametrů, liší se porovnávané produkty jen v detailech. Tato práce však nezjišťuje další nastavení jednotlivých vlastností virtualizačních řešení, jako jsou rychlost, spolehlivost, dostupnost a další. Tyto vlastnosti jsou většinou odvislé od konkrétního nasazení a mohou se tak velmi odlišovat. Sledováno je tak pouze zda daný produkt technologii obsahuje nebo ne.

### 5.3.1 Vlastnosti hypervizoru

| | ESX | XenServer | Hyper-V |
|---|---|---|---|
| Typ CPU | 64 bit | 64 bit | 64 bit |
| Hardwarová podpora virtualizace | Není nutná | Nutná pro Windows | Nutná vždy |
| Maximální počet jader | 64 | 32 | 64 |
| Rozsah paměti | 2 GB – 1 TB | 1 GB – 128 GB | 1 GB – 1 TB |

Jak je vidět, jediný zásadní rozdíl je v nutnosti hardwarové podpory virtualizace Intel-VT nebo AMD-V. Dalším rozdílem je omezení počtu jader u hypervizoru Xen. Rozsah paměti, nutný, respektive povolený, k provozu fyzického hostitele je dostatečně velký, aby ani v jednom případě nehrál zásadní roli.

### 5.3.2 Management tools

| | ESX | XenServer | Hyper-V |
|---|---|---|---|
| Podpora Clusteru | 32 | 32 | 32 |
| High Availability | Ano | Ano | Ano |
| DRS | Ano | Ne | Ne |
| Live Migration | Ano | Ano | Ano |
| Storage Migration | Ano | Ano | Ano |
| Fault Tolerance | Ano | Ano | Ano |

Z hlediska Management Tools je jediný podstatný rozdíl v podpoře technologie Dynamic Resource Scheduler. Tato technika umožňuje dynamické přidělování zdrojů dle potřeby hostovaného systému. Výhoda této technologie spočívá v možnosti přetěžování zdrojů, kdy maximální možný dostupný zdroj všech hostovaných systémů dohromady převyšuje fyzické dispozice hostujícího systému. Vzhledem k tomu, že se však výkonová špička jednotlivých hostovaných strojů v čase liší, pravděpodobnost, že dojde k nedostatku zdrojů je poměrně malá.

### 5.3.3 Networking

Z hlediska síťových služeb se jednotlivé produkty opět téměř neliší. Všechny obsahují následující produkty:

- Podpora VLAN
- Podpora Virtual Switch
- Podpora IPv6

Jediné, v čem se různí, je podpora Cisco Discovery Protocol. Ten podporuje, díky smlouvě o spolupráci se společností Cisco, pouze produkt VMware ESX Server.

Zajímavostí je, že technologie Virtual Switch byla pro produkt Hyper-V převzata, opět díky vzájemným dohodám, z produktu XenServer.

### 5.3.4 Storage

Stejně jako v předchozích případech, i zde jsou rozdíly poměrně malé. Pomineme-li, že produkt ESX neobsahuje podporu pro disková zařízení na sběrnici PATA (která se již v serverových řešeních stejně vyskytuje pouze zřídka), mají všechny produkty podporu standardů SATA, SCSI a SAS. Stejně tak je ve všech produktech implementována podpora iSCSI a Fibre Channel. Jediný rozdíl je tak v podporovaných Cluster File Systémech, kde ESX podporuje standard VMFS, Hyper-V standard CSV/NTFS a XenServer zatím stále na pořádnou implementaci Cluster File Systému čeká.

### 5.3.5 Výkonnost hypervizorů

Předchozí srovnání „papírových“ parametrů vybraných virtualizačních technik neříká nic o jejich výkonu v reálném nasazení. Sice již bylo řečeno, že výkon jednotlivých technik je silně závislý na jednotlivých implementacích a v rámci různých realizací se liší, nicméně alespoň hrubá představa o výkonu jednotlivých virtualizérů je důležitá.

Autoři serveru virtualizationreview.com (VirtualizationReview, 2009 [15]) si položily otázku, zda existuje výrazný rozdíl ve výkonu jednotlivých hypervizorů. Rozhodli se tedy otestovat, jak si jednotlivá virtualizační řešení stojí. Vytvořili stres test, který hodnotil za jinak stejných podmínek výkon hypervizorů VMware ESX, Microsoft Hyper-V a Citrix Xen. V rámci tohoto testu porovnávali pouze výkon jednotlivých hypervizorů a jejich schopnosti. Neřešili tedy další přidané funkce, jako jsou migrace, tolerance k chybám, přidělování zdrojů a další součásti Management Tools.

Stres test byl koncipován tak, aby pokud možno kopíroval předpokládanou běžnou zátěž hypervizorů. Jako hostující stroje bylo instalováno množství Windows Serverů 2003, z nichž na jednom běžel Microsoft SQL Server 2005. Všechny stroje byly pouze 32bitové, aby byla zajištěna naprostá kompatibilita se všemi testovanými hypervizory.

Celkově byly prováděny 3 testy:

- Malé množství výpočetně náročných systémů: obsahoval 1 databázový server se středně velkou databází a 6 virtuálních strojů, které spuštěnými procesy silně vytěžovaly procesor a paměť
- Velké množství výpočetně náročných systémů: obsahoval 1 databázový server se středně velkou databází a 12 virtuálních strojů, které spuštěnými procesy silně vytěžovaly procesor a paměť
- Velké množství výpočetně nenáročných systémů: obsahoval 1 databázový server se středně velkou databází a 12 virtuálních strojů, které spuštěnými procesy lehce vytěžovaly procesor a paměť.

Pro měření byl použit program PassMark a jako měřící jednotky byly použity následující:

- CPU – počet matematických operací s plovoucí čárkou za hodinu
- Disk – počet přečtených a zapsaných bloků o velikosti 32KB s využitím 15% kapacity diskového prostoru
- RAM – počet operací, které v pěti krocích zaplnily a opět uvolnily operační paměť
- SQL – provedení definované operace nad databázovým systémem, měřené v čase s využitím metod měření času implementovaných v databázi

Výsledky jednotlivých hypervizorů jsou uvedeny v příloze číslo 1.

Z výsledků je zřetelné, že testované hypervizory se ani v konkrétním stres testu mnoho neliší. Jedinou výjimku tvoří překvapivě VMware ESX 3.5, který zaostává v případě vyšší zátěže. I z výsledků je patrné, že se spíše hodí do prostředí, kde je používáno větší množství na výkon nenáročných strojů.

Jak již ale bylo několikrát řečeno, nelze srovnávat pouze na základě výkonu samotných hypervizorů, jelikož přidaná hodnota každého virtualizačního řešení spočívá především v implementaci dodatečných služeb tím kterým výrobcem. Proto konkrétní výběr toho kterého produktu záleží především na účelu nasazení, požadovaných službách a licenčních podmínkách.

# 6 Realizace virtuální serverové infrastruktury ve společnosti Atelier Slanec s.r.o.

Společnost Atelier Slanec s.r.o. je architektonický atelier, který se zaměřuje na architektonickou a projektovou přípravu staveb nebo větších stavebních celků a autorský dozor při jejich realizaci. Jedná se o menší společnost, která zaměstnává cca 10 stálých zaměstnanců a řadu externistů. I přesto má poměrně velké portfolio klientů, kterým poskytuje své služby.

Informační technologie ve společnosti jsou využívány standardně k realizaci pracovní činnosti a podpoře zákazníků. K tomuto účelu je sídlo společnosti Atelier Slanec s.r.o. vybaveno standardní kancelářskou IT infrastrukturou, která obsahuje prvky relevantní k zaměření společnosti.

## *6.1 Výchozí stav*

Z hlediska serverové infrastruktury jsou využívány dva servery, které zabezpečují veškeré serverové operace. První server je výkonný stroj, který slouží k náročným grafickým operacím, které jsou nezbytné pro provoz společnosti (renderování architektonických návrhů a podobně). Tento server je vzhledem ke svému účelu natolik důležitým prvkem informační architektury společnosti, že bylo rozhodnuto ponechat jej tak jak je a v následujícím textu tak není uvažován. Druhý stroj zajišťuje veškeré další serverové operace. Vzhledem k tomu, že zbývající operace nejsou nikterak náročné na výpočetní výkon, byl pro tyto účely kdysi vyhrazen starší stroj, který i v dnešní době s dostatečnými rezervami zajišťuje požadované operace. Jedná se především o soubor následujících činností:

- Vstupní bod do firemní sítě (router, firewall)
- Webhostingový server pro klientský a zaměstnanecký přístup
- Sdílené úložiště (Samba server)
- Databázový zdroj pro interní programy (SQL server)

Jak již bylo uvedeno, žádná z požadovaných služeb není extrémně výpočetně náročná. Veškeré služby jsou zajišťovány operačním systémem Linux, konkrétně distribucí

Ubuntu 9.4 v serverové variantě. Bezproblémový běh služeb je realizován na následující konfiguraci:

- Intel Pentium Dual-Core 1,3 GHz
- 3 GB RAM
- 2x SATA 1TB v RAID 1
- 1x 10/100/1000 Mb Ethernet card
- 1x 10/100 Mb Ethernet card

Ostatní parametry (grafická karta, motherboard, …) nejsou z hlediska použití podstatné.

## 6.2 Žádaný stav

Ve výchozím stavu server splňoval veškeré požadavky, které na něj byly kladeny. Jeho provoz byl, až na pár incidentů, v podstatě bezúdržbový. Proč tedy tento stav měnit? Odpověď na předchozí otázku je především bezpečnost. Vzhledem k tomu, že server sloužil jako vstupní bod do firemní infrastruktury a zároveň jako databázový a souborový server, který obsahoval citlivá firemní data, je otázka zabezpečení na místě. V historii společnosti se sice ještě žádný pokus o proniknutí do tohoto serveru nestal, potenciálně však toto riziko může hrozit.

Dalším důvodem, proč změnit výchozí stav je pohled do budoucnosti, kdy může vyvstat potřeba převézt některou z poskytovaných serverových činností na výkonnější platformu. V současném stavu je oddělení jednotlivých činností díky jejich poměrně malé kapacitě poměrně snadné. Vzhledem k tomu, že k jednotlivým službám jsou v čase přidávány další funkcionality (týká se především rozšiřování SQL databáze a klientského webu), mohlo by být toto oddělení v budoucnu složité. Stejně tak je pravděpodobné, že v budoucnu přibudou ke stávajícím serverovým službám další.

V rámci budoucího stavu serverové infrastruktury je tedy potřeba vyřešit především následující problémy:

- Bezpečnost
- Škálovatelnost
- Přenositelnost

- Efektivní využití zdrojů

## 6.3 Výběr řešení

V rámci diskuze o možnostech docílení žádaného stavu byly navrhnuty 2 možné varianty:

**Zakoupení dalších serverů**

Tato varianta by obnášela poměrně velké množství investic. Jednalo by se především o pořizovací investice, jelikož další vyřazené stroje již nejsou k dispozici. Dále by tato varianta vyžadovala další náklady určené na provoz. Nejasné také bylo řešení umístění další výpočetní techniky v prostorách společnosti (při zachování dostupnosti energetických a dalších požadavků). Na druhou stranu by tato varianta přinášela dodatečnou kapacitu ve výpočetním výkonu a i v případě prudkého nárůstu požadavků na výpočetní výkon by obstála.

**Virtualizace na současném serveru**

Varianta virtualizace sice poměrně zdárně řeší veškeré nevýhody předcházející varianty, nastoluje ovšem další problémy. Především je to složitost její implementace, kdy je potřeba mimo oddělení jednotlivých serverových služeb, zajistit implementaci virtualizačních technik. Dále je to existence poměrně starého, i když stále postačujícího, fyzického zařízení. Na druhou stranu však tato varianta neplýtvá výpočetním výkonem a do budoucna je připravena na případné rozšiřování.

Po úvaze a diskuzi s majitelem společnosti bylo rozhodnuto, že se přistoupí k virtualizaci serverové infrastruktury. V rámci pilotního projektu pak bylo rozhodnuto, že pokud to nebude nezbytně nutné, nebude se dokupovat nové vybavení a realizace bude probíhat na současném fyzickém hardware.

Toto rozhodnutí však celkový proces virtualizace poměrně zkomplikovalo. V konečném důsledku se jednalo především o značné zúžení virtualizačních produktů, které připadají v úvahu. Toto zúžení bylo způsobeno především skutečností, že stávající

hardware je staršího data a nepodporuje mnohé z v současnosti využívaných virtualizačních technik. V rámci výběru vhodného virtualizačního produktu tak musela být tato skutečnost zohledněna. Po analýze trhu bylo vybráno několik produktů, které jsou k danému účelu vhodné:

- VMware Server 2 (dříve VMware GSX Server)
- Xen hypervizor
- VServer

Další produkty byly z řady důvodů vyloučeny, jelikož vyžadovaly buď přímo běh na dedikovaných serverech (VMware ESX), hardwarovou podporu virtualizace (KVM) nebo nebyly vhodné k běhu serverů založených na operačním systému Linux (Hyper-V).

V následujících krocích byl z výběru vyřazen produkt VServer. Tento produkt neposkytuje úplnou virtualizaci a pouze partitioning (viz. Kapitola 5.2). To by přinášelo některé výhody (snížení nároků na výkon), ale z pohledu do budoucnosti značně omezilo možnosti migrace a škálovatelnosti. Rozhodnutí nakonec padlo na produkt VMware Server 2 a to hlavně díky potenciální podpoře hostování operačního systému Windows. Tato podpora sice v současné době není využitelná, ale do budoucna směřují z různých důvodů úvahy k jejímu nasazení. Dále byla zohledněna bezproblémová potenciální migrace hostovaných strojů do novějších produktů společnosti VMware.

## *6.4 Implementace*

### 6.4.1 Tvorba virtualizační infrastruktury

V rámci implementace bylo shledáno několik poměrně závažných problémů, které celý proces znesnadnily. Tyto problémy většinou vyplývaly ze stáří produktu VMware Server 2, jež již není některými novějšími distribucemi operačních systémů podporován. První problém se týkal samotné instalace, druhý se dotýkal především práce s VMware Infrastructure Web Access, přes který je možné spravovat virtuální prostředí. Konkrétní problémy a jejich řešení je uvedeno dále v textu.

Vzhledem k tomu, že produkt VMware server 2 vyžaduje ke svému běhu hostitele, bylo v rámci realizace potřeba nejprve nainstalovat hostitelský systém. Vzhledem k tomu,

že existující infrastruktura byla založená na linuxové distribuci Ubuntu Server a její prostředí bylo zodpovědným osobám ve společnosti Atelier Slanec s.r.o. známo, padla volba opět na tuto distribuci v poslední verzi. Popis instalace hostitelského systému svým zaměřením přesahuje tuto práci, tedy zde nebude dále upřesňován.

Po samotné instalaci hostitelského systému a nastavení důležitých, především síťových, služeb na tomto systému bylo přistoupeno k instalaci samotného virtualizačního produktu. Zde se vyskytla první komplikace, jelikož od Ubuntu verze 9.04 již VMware server není v této distribuci podporován. Tato skutečnost je způsobena ukončeným vývojem produktu VMware Server 2 a některými inovacemi jádra operačního systému linux, které ve stávající verzi VMware Server 2 nepodporuje. Jedná se především o kompilaci a zavedení některých nezbytných modulů do jádra v průběhu instalace (především moduly vmmon, vmci a vsock). Tento problém se podařilo odstranit s přispěním Rada Cotescua, který na svém blogu (Cotescu, 2009 [16]) uveřejnil skript, který s úspěchem tyto problémy řeší.

Po instalaci samotného VMware Server 2 bylo nutné provézt jeho konfiguraci. Ta probíhá interaktivně pomocí konfiguračního skriptu *vmware-config.pl*, který je součástí instalačního balíku. V rámci konfigurace se vyplňují informace o uložení jednotlivých virtuálních strojů, konfiguruje se administrační rozhraní VMware Server 2, porty na kterých bude webové konfigurační rozhraní naslouchat, oprávnění uživatelů a další. Po této konfiguraci byl již produkt VMware Server 2 schopen běhu.

Další problém však nastal v souvislosti s webovým administračním rozhraním VMware Infrastructure Web Access. Jeho součástí je plug-in VMware Remote Console Plug-in, který je však podporován pouze v některých internetových prohlížečích a jen v některých jejich verzích. Pro jeho používání byl z rozhodnutí správce infrastruktury a z uživatelského hlediska vybrán prohlížeč Mozilla FireFox, který však musel být downgradován na verzi 3.5, jelikož vyšší verze nepodporují výše zmíněný plug-in.

### 6.4.2 Tvorba jednotlivých serverů

Z pohledu tvorby jednotlivých virtuálních serverů bylo důležité především správné nastavení jejich parametrů. Vzhledem k omezením, daným dostupnými fyzickými zdroji,

byl tento krok maximálně důležitý. Po důkladné analýze bylo rozhodnuto následující přidělení zdrojů:

| | RAM | Velikost disku | Síťová karta |
|---|---|---|---|
| Router, firewall | 256 MB | 10 GB | 2x bridge |
| Webhosting | 512 MB | 10 GB | 1x bridge |
| Samba server | 512 MB | 10 GB + 500 GB | 1x bridge |
| SQL server | 1024 MB | 100 GB | 1x bridge |

Přidělení jednotlivých zdrojů bylo plánováno s ohledem na předpokládané využití. Výhodou virtuální infrastruktury je, že v případě potřeby lze jednotlivá nastavení poměrně bezproblémově měnit. Vzhledem k tomu, že byl k dispozici dvoujádrový procesor, bylo možné jedno jádro vyhradit do užívání SQL serveru. Ostatní servery využívaly druhé jádro procesoru. K existenci 2 diskových oddílů v případě souborového Samba serveru bylo přistoupeno z důvodu praktičnosti oddělení systémového disku virtuálního stroje a samotných distribuovaných dat. Hostitelský počítač byl dále doplněn o 4 síťové karty, které byly rozděleny mezi jednotlivé virtualizované stroje. Jedna síťová karta pak byla vyhrazena hostitelskému počítači a to především z důvodu možnosti administrace virtuálního rozhraní. Veškeré síťové karty jsou v režimu bridge, což znamená, že jsou přímo namapována na fyzické rozhraní hostitelského počítače. To je realizováno především z toho důvodu, že všechny servery poskytují zdroje, které musí být dostupné v síťovém provozu. Z hlediska zrychlení výměny dat mezi některými servery (webhosting, SQL) byla v počátečním návrhu uvažována ještě jedna virtuální síť přímo mezi virtualizovanými servery. Od této realizace bylo prozatím upuštěno, ale její implementace by byla v současném stavu poměrně jednoduchá.

Samotná instalace serverů a nastavení jejich služeb opět přesahuje rámec této práce. Jedna věc však z hlediska zavádění virtuální infrastruktury musí být uvedena, jelikož se jedná o poměrně zásadní usnadnění celého procesu. Vzhledem k tomu, že instalace všech virtuálních serverů byla v základu stejná a každý server se od těch ostatních lišil až doinstalovaným programovým vybavením a spuštěnými službami, byla instalace každého

ze serverů odvozena z jedné instalované instance. V praxi byl tento krok realizován instalací jednoho serveru, kterému byly nastaveny parametry společné všem virtuálním serverům. Tento obraz byl následně zduplikován a z těchto kopií původního systému pak byly, po úpravě některých parametrů, vytvořeny zbývající servery. Tato skutečnost znamenala poměrně výraznou úsporu času, jelikož nebylo nutné každý server instalovat zvlášť, ale stačilo nainstalovat pouze jeden a z něj vytvořit zbývající.

Po nastavení jednotlivých serverů a jejich konfiguraci bylo nutné zajistit jejich automatické spuštění po zapnutí fyzického zařízení. Předpokládá se sice, že fyzický server, stejně jako virtuální servery, bude běžet nonstop, v případě dlouhodobých výpadků proudu či jiných nestandardních okolností by však bylo nepříjemnou nutností spouštět každý virtualizovaný systém zvlášť. Automatické spuštění virtuálních serverů po startu hostitele bylo zajištěno standardní cestou přes rc.d skripty, ke kterým bylo přiřazeno spouštění několika příkazů zajišťujících spuštění virtuálních strojů. Obecná podoba spouštěcího příkazu je:

```
/usr/bin/vmrun –u uživatel –h 'http://localhost:8222/sdk' –p heslo start [datové úložiště] cesta/k/virtuálnímu/serveru
```

V příkazu je jasně definován uživatel a heslo, které zajišťuje potřebná oprávnění ke spuštění virtuálního serveru. V případě automatického spuštění je tato ochrana víceméně zbytečná, v případě manuálního spouštění přes příkazový řádek se tato ochrana hodí. Parametr -h určuje cestu ke sdíleným knihovnám VMware serveru 2. Vzhledem k tomu, že tento produkt v sobě má instalovaný vlastní webový server, který zajišťuje administraci virtuálního prostředí, je směrování pomocí tohoto formátu nejjednodušší. Podmínkou je, aby se tento příkaz prováděl až po zavedení všech součástí VMware serveru. Datové úložiště pak definuje použitý datastore, který obsahuje virtuální stroj. Vzhledem k tomu, že produkt VMware Server může obsahovat více datových úložišť, je tato identifikace nezbytná.

### 6.4.3 Automatizace zálohování

Problematika zálohování obsahu jednotlivých virtuálních serverů již byla součástí původního řešení. Veškerá důležitá data jsou v rámci infrastruktury zrcadlena v diskovém

poli, což samo o sobě poskytuje poměrně efektivní způsob ochrany proti poruše některého z disků. Vzhledem k tomu, že data jsou nejcennějším aktivem společnosti Atelier Slanec s.r.o., a vlastně jakékoliv společnosti, bylo již u původního řešení přistoupeno k pravidelnému zálohování těchto dat do geograficky odlišné lokality (konkrétně na server u majitele společnosti doma). Z hlediska nové, virtuální, infrastruktury stačilo proces zálohování pouze drobně upravit, což bylo součástí konfigurace jednotlivých virtualizovaných strojů.

V rámci virtualizace infrastruktury však bylo nutné zajistit též pravidelné zálohování běžících virtuálních strojů jako celku. Naneštěstí utility pro automatizaci záloh pomocí pravidelného snímkování virtuálních strojů jsou doménou pouze robustnějších a placených produktů společnosti VMware. Produkt VMware Server 2 však sám o sobě poskytuje možnost vytvoření snímku běžícího virtuálního stroje. Tato možnost je však omezena pouze na vytvoření jednoho snímku. Při vytvoření dalšího je již existující snímek přepsán. Proto byl vytvořen skript, který umožňuje pravidelné vytváření snímků a jejich zálohu na odlišné místo v rámci souborového systému hostitelského počítače. Vzhledem k tomu, že již při návrhu fyzických parametrů jednotlivých strojů bylo s tímto modelem počítáno, existuje v rámci hostitelského počítače dostatečná kapacita na uložení 3 v čase různých snímků virtuálních strojů. Provádí se záloha pouze stavu virtuálních strojů a jejich primárních disků. Nepřistupuje se tak k záloze veškerých dat souborového Samba serveru, jelikož tato jsou zálohována jiným způsobem a vytvářením snímku disku, na němž jsou uložena, by způsobovalo nadbytečnou redundanci a bylo extrémně časově náročné.

Samotný skript je pak tvořen s ohledem na univerzalitu a možnost přidání dalších zálohovaných strojů při jejich případném vytvoření. Samotný skript vznikl úpravou původního skriptu ghettoVCB ESX(i), který byl vytvořen Wiliamem Lamem za účelem vytváření záloh virtualizačního produktu VMware ESX(i) Server a jeho modifikací, kterou vytvořil Michael Dixon.

Veškeré virtualizační produkty společnosti VMware je možné ovládat z příkazové řádky. Použitý skript tak pouze využívá funkcí, které produkt VMware Server 2 poskytuje. Veškeré funkce poskytuje rozhraní VMware CLI management utility, které je voláno prostřednictvím programu /usr/bin/vmware-vim-cmd. Tento program umožňuje jak základní ovládání virtuálních strojů (spouštění, vypínání, …), tak jejich pokročilý management

(přidělování zdrojů, vytváření obrazů, …). Příkaz pro vytvoření virtuálního obrazu je následující:

```
/usr/bin/vmware-vim-cmd vmsvc/snapshot.create ${VM_ID} vcb_snap VCB_BACKUP_${VM_NAME}_`date +%F`
```

Proměnná VM_ID zde identifikuje příslušný virtuální stroj, jehož záloha bude vytvořena do umístění VCB_BACKUP_jméno stroje_datum vytvoření. Následně je nutné tuto zálohu přemístit do umístění určeného administrátorem. Přesunutí je důležité z toho důvodu, že při případném vytvoření další zálohy by byla stávající záloha přepsána. Tento přesun je realizován pomocí nástroje rsync, který je dnes většinou základní výbavou linuxových systémů. Jeho syntaxe je v tomto případě následující:

```
rsync -av --progress --exclude=*-000001*.vmdk ${VMX_DIR}/*.vmdk "${VM_BACKUP_DIR}/"
```

Zbývající části skriptu zajišťují především podpůrnou funkcionalitu, pomocí které jsou zjišťovány nezbytné informace, mazány staré zálohy a ošetřeny chyby.

## 6.5 Zhodnocení

V rámci realizace virtuální serverové infrastruktury ve společnosti Atelier Slanec s.r.o. bylo vytvořeno stabilní virtuální prostředí s využitím minima dodatečných prostředků v porovnání s výchozím stavem. Za cenu určitých komplikací při zavádění nového systému (především dočasná nedostupnost některých služeb) bylo získáno prostředí, které se v porovnání s výchozím stavem vyznačuje především následujícími vlastnostmi:

**Vyšší bezpečnost**

Oddělením služeb, provozovaných původně jedním serverem, bylo docíleno výrazně vyššího zabezpečení v porovnání s výchozím stavem. Prolomení nebo chyba jednoho ze systému již nemůžou ohrozit funkcionalitu ostatních.

**Škálovatelnost**

V porovnání s výchozím řešením je přidělování zdrojů jednotlivým strojům mnohem efektivnější. Odhlédneme-li od faktu, že přidělení prostředků pouze jedné

službě bylo ve výchozím stavu nemožné, získali jsme plně škálovatelné prostředí, ve kterém není problém v závislosti na vytížení dodávat či ubírat jednotlivé zdroje.

**Přenositelnost**

Vzhledem k tomu, že v rámci vytvoření jednotlivých virtuálních strojů bylo potřeba přenést původní data do nových systémů, dalo by se říci, že již původní systém byl přenositelný. V současné době je však systém plně připraven na případnou migraci na jiného fyzického hostitele a to jak v případě poruchy současného fyzického zařízení, tak v případě potřeby zvýšení výkonu.

Samozřejmě se v průběhu několika týdnů po zavedení virtuální infrastruktury vyskytlo určité množství problémů. Ty však byly způsobeny především nastavením služeb běžících v jednotlivých virtuálních serverech, které v některých parametrech neodpovídaly původnímu nastavení. Tyto problémy se však netýkaly samotné virtuální infrastruktury a daly se většinou poměrně rychle vyřešit.

# 7 Závěr

V rámci této práce jsem se snažil pokud možno co nejsrozumitelněji vymezit pojem virtualizace. Nejprve jsem vysvětlil pojem hypervizor a definoval nezbytné podmínky virtualizace, jak je vyjádřili pánové Popek a Goldberg. Následně jsem popsal obecné možnosti virtualizace procesoru, paměti a vstupně-výstupních zařízení. Pouze poté mohla následovat komparace nejpoužívanějších virtualizačních technik z hlediska přístupu k dostupným zdrojům, efektivitě a náročnosti realizace.

V další části jsem se snažil čtenáře seznámit s trhem virtualizačních produktů. S přihlédnutím k velikosti trhu a tržním podílům jednotlivých výrobců jsem popsal vybrané společnosti a jejich produkty. Vybral jsem tři nejčastěji nasazované produkty a s pomocí již existujících analýz provedl jejich srovnání.

V rámci praktické části této práce jsem pak demonstroval nasazení virtualizace v konkrétním prostředí. Tato činnost pro mne byla velkou zkušeností, při které jsem si mohl ověřit některé z teoretických poznatků v praxi. Největší uspokojení však přinesl fungující systém, který se vyznačoval všemi požadovanými vlastnostmi.

Osobně doufám, že všechny zde uvedené informace byly čtenáři přínosem a přečtení práce mu pomohlo k lepší orientaci v dané problematice. Nutno však podotknout, že ve zdrojích z kterých jsem čerpal je zajímavých informací mnohem více.

# 8 Seznam literatury


1. Mitch Tulloch. Understanding Microsoft Virtualization Solutions. Redmond: Microsoft Press, 2009, 417 s. ISBN 9780735693371
2. Marshall, D. – Reynolds, W.A. – McCrory, D. Advanced server virtualization: VMware and Microsoft platforms in the virtual data center. Boca Raton: Auerbach, 2006. 742 s. ISBN 0-8493-3931-6
3. Lowe, W.J. VMware Infrastructure 3 for Dummies. Hoboken: Wiley Publishing, Inc., 2008. 316 s. ISBN 978-0-470-27793-5
4. Jerrod, B. VMware 100 Success Secrets – 100 Most Asked Questions: The Missing VMware Server, Workstation Planning, Installation and Management Introduction Guide. Emereo Pty Ltd, 2008. 160 s. ISBN 978-1-921523-02-1
5. L. Matyska. Techniky virtualizace počítačů (2). Zpravodaj ÚVT MU. ISSN 1212-0901, 2007, roč. XVII, č. 3, s. 9-12. Dostupný na www: *http://www.ics.muni.cz/zpravodaj/articles/545.html*
6. Ruest, D. – Ruest, N. Virtualizace Podrobný průvodce. Computer Press, 2010. 408 s. ISBN 978-80-251-2676-9
7. Robin, J.S., Irvine, C.E. Analysis of the Intel Pentium's ability to support a secure virtual machine monitor. In SSYM'00: Proceedings of the 9th conference on USENIX Security Symposium, Berkeley, CA, USA, 2000. USENIX Association.
8. Děcký, M. Mechanismy virtualizace běhu operačních systémů – Diplomová práce. Katedra softwarového inženýrství Matematicko-fyzikální fakulta Univerzita Karlova v Praze, 2006. 68 s.
9. Patka, L. Virtualizační technologie – Diplomová práce. Fakulta informatiky Masarykova univerzita v Brně, 2009. 64 s.
10. Adams, K., Agesen, O. A comparison of software and hardware techniques for x86 virtualization. In ASPLOS, 2006.
11. Bittman, T. J. – Dawson, P. – Weiss, J. Magic Quadrant for x86 Server Virtualization Infrastructure. Gartner Inc., 2010
12. Neiger, G., Santoni, A., Leung, F., Rodgers, D., Uhlig, R. Intel○R Virtualization Technology: hardware support for efficient processor virtualization. Intel Technology Journal, 10(3):167–177, 2006.
13. Popek, G.J., Goldberg, R.P. Formal requirements for virtualizable third generation architectures. Commun. ACM, 17(7):412–421, 1974.

14. Wolf, Ch. Hypervisor Competitive Differences: Beyond the Data Sheet, 2009.
15. VirtualizationReview – Lab Experiments Hypervisors http://virtualizationreview.com/Articles/2009/03/02/Lab-Experiment-Hypervisors.aspx?Page=1
16. Radu Cotescu - personal webpages – How to instar vmware server 2.0 on Ubuntu 9.10 http://radu.cotescu.com/how-to-install-vmware-server-2-0-x-on-ubuntu-9-10-karmic-koala/
17. eArchiv.cz – archiv článků a přednášek Jiřího Peterky - Simulace vs. Emulace http://www.earchiv.cz/a92/a210c120.php3
18. VMware.com http://www.vmware.com
19. VMwarenews.cz - VMware nyní nabízí hypervisor ESXi zdarma http://www.vmwarenews.cz/vmw/vmwnews.nsf/0/3E59DB93AE24E1E3C12574CA004EF3AC
20. Goldberg, R. P. Architectural Principles for Virtual Computer Systems. Harvard University, 1973. 229 s.

# 9 Přílohy

## Příloha 1

Výsledky jednotlivých stres-testů vybraných hypervizorů. Zdroj [15]

**Table 2. Microsoft Hyper-V Performance Results**

| | |
|---|---|
| **TEST 1** | |
| Number of workload virtual machines: | 6 |
| Number of database virtual machines: | 1 |
| Average CPU operations per VM: | 29 billion per hour |
| Average RAM operations per VM: | 108.83 billion per hour |
| Average SQL job completion time: | 4 minutes 40 seconds |
| **TEST 2** | |
| Number of workload virtual machines: | 12 |
| Number of database virtual machines: | 1 |
| Average CPU operations per VM: | 8.67 billion per hour |
| Average RAM operations per VM: | 31.75 billion per hour |
| Average Disk operations per VM: | 416 million per hour |
| Average SQL job completion time: | 6 minutes |
| **TEST 3** | |
| Number of workload virtual machines: | 12 |
| Number of database virtual machines: | 1 |
| Average CPU operations per VM: | 5 billion per hour |
| Average RAM operations per VM: | 1.08 billion per hour |
| Average Disk operations per VM: | 167 million per hour |
| Average SQL job completion time: | 4 minutes 43 seconds |

**Table 3. Citrix XenServer Performance Results**

| | |
|---|---|
| **TEST 1** | |
| Number of workload virtual machines: | 6 |
| Number of database virtual machines: | 1 |
| Average CPU operations per VM: | 36.5 billion per hour |
| Average RAM operations per VM: | 112 billion per hour |
| Average SQL job completion time: | 4 minutes 26 seconds |
| **TEST 2** | |
| Number of workload virtual machines: | 12 |
| Number of database virtual machines: | 1 |
| Average CPU operations per VM: | 8.75 billion per hour |
| Average RAM operations per VM: | 26.67 billion per hour |
| Average Disk operations per VM: | 583 million per hour |
| Average SQL job completion time: | 8 minutes 3 seconds |
| **TEST 3** | |
| Number of workload virtual machines: | 12 |
| Number of database virtual machines: | 1 |
| Average CPU operations per VM: | 3.75 billion per hour |
| Average RAM operations per VM: | 1.25 billion per hour |
| Average Disk operations per VM: | 187 million per hour |
| Average SQL job completion time: | 5 minutes 34 seconds |

**Table 4. VMware ESX 3.5 Performance Results**

| | |
|---|---|
| **TEST 1** | |
| Number of workload virtual machines: | 6 |
| Number of database virtual machines: | 1 |
| Average CPU operations per VM: | 29.5 billion per hour |
| Average RAM operations per VM: | 88.5 billion per hour |
| Average SQL job completion time: | 5 minutes 8 seconds |
| **TEST 2** | |
| Number of workload virtual machines: | 12 |
| Number of database virtual machines: | 1 |
| Average CPU operations per VM: | 3.67 billion per hour |
| Average RAM operations per VM: | 10 billion per hour |
| Average Disk operations per VM: | 667 million per hour |
| Average SQL job completion time: | 11 minutes 28 seconds |
| **TEST 3** | |
| Number of workload virtual machines: | 12 |
| Number of database virtual machines: | 1 |
| Average CPU operations per VM: | 7.08 billion per hour |
| Average RAM operations per VM: | 1.25 billion per hour |
| Average Disk operations per VM: | 187 million per hour |
| Average SQL job completion time: | 5 minutes 34 seconds |

## Příloha 2

Zdrojový kód skriptu použitého k pravidelnému automatickému vytváření záloh virtuálních serverů.

```
#!/bin/bash
#-------------------------------------------------------------
# ZALOHOVANI VIRTUALNICH STROJU
# vytvoreno upravami skriptu ghettoVCB ESX(i) od
# Williama Lama, upraveneho Michaelem Dixonem
#-------------------------------------------------------------
shopt -s extglob
#-------------------------------------------------------------
# Nastaveni udaju pro autentifikaci
#-------------------------------------------------------------
VM_USER=
VM_PASSWORD=
#-------------------------------------------------------------
# Pridani moznosti zabaleni zalohy do archivu
#-------------------------------------------------------------
ENABLE_ARCHIVING=0
#-------------------------------------------------------------
# Umisteni zalohy
#-------------------------------------------------------------
VM_BACKUP_VOLUME=
#-------------------------------------------------------------
# Pocet udrzovanych zaloh pred smazanim
#-------------------------------------------------------------
VM_BACKUP_ROTATION_COUNT=3
#-------------------------------------------------------------
# Zpusob pojmenovani adesaru se zalohou
#-------------------------------------------------------------
VM_BACKUP_DIR_NAMING_CONVENTION="$(date +%F)"
#-------------------------------------------------------------
# Provadeni zalohy pri vypnutem virtualnim stroji
# Vypinani pomoci VMtools
#-------------------------------------------------------------
POWER_VM_DOWN_BEFORE_BACKUP=0
#-------------------------------------------------------------
# V pripade absence VMTools vypnout "natvrdo"
#-------------------------------------------------------------
ENABLE_HARD_POWER_OFF=1
#-------------------------------------------------------------
# Prodleva pred tvrdym vypnutim
#-------------------------------------------------------------
ITER_TO_WAIT_SHUTDOWN=4
#-------------------------------------------------------------
# Samotne telo skriptu
#-------------------------------------------------------------
DEBUG_MODE=0
# Verze skriptu
VERSION="1.5"
#-------------------------------------------------------------
# Funkce : debugVar
#-------------------------------------------------------------
debugVar() {
  if [ ${DEBUG_MODE} -eq 1 ]; then
                echo -e "DEBUG: $1 = $2"
  fi
}
#-------------------------------------------------------------
# Funkce : printUsage
#-------------------------------------------------------------
printUsage() {
                SCRIPT_PATH=$(basename $0)
                echo          -e          "\nUsage:          ${SCRIPT_PATH}
[VM_FILE_INPUT]\n"
}
#-------------------------------------------------------------
# Funkce : getProduct
#-------------------------------------------------------------
getProduct() {
  getVimVar "hostsvc/hostsummary" productLineId
}
#-------------------------------------------------------------
# Funkce : getVimVar
#-------------------------------------------------------------
getVimVar() {
  debugVar "Searching $1 for" "$2"
  local CMD="${VMWARE_CMD} $1"
  ASSIGN_VAR=`eval $CMD | grep "$2" | awk -F " = " '{print $1 "="
$2}' | sed 's/,//g'`
  if [ "${ASSIGN_VAR}" != "" ]; then
    eval $ASSIGN_VAR
  fi
  if [ ! -z $3 ]; then
    debugVar "Alternate variable specified" "$3"
    ASSIGN_VAR="$3=\$2"
    eval $ASSIGN_VAR
    unset $2
  fi
}
#-------------------------------------------------------------
# Funkce : getSnapshotTasks
#-------------------------------------------------------------
getSnapshotTasks() {
  debugVar "Getting Snapshot Tasks for" "${VM_ID}"
  local ST_IFS=$IFS
  IFS=":"
  local CMD="${VMWARE_CMD} vmsvc/get.tasklist ${VM_ID}"
  SNAPSHOT_TASKS=`eval $CMD | grep createSnapshot | awk -F
":" '{print $2}' | sed s/"""//g | sed 's/,//g'`
  IFS=$ST_IFS
}
#-------------------------------------------------------------
# Funkce : get SnapshotStatus
#-------------------------------------------------------------
getSnapshotStatus() {
  debugVar "Getting Snapshot State for" "$1"
  getVimVar "vimsvc/task_info $1" state
}
#-------------------------------------------------------------
# Funkce : checkVMBackupRotation
#-------------------------------------------------------------
checkVMBackupRotation() {
  IFS=$'\012'
                local BACKUP_DIR_PATH=$1
                local BACKUP_VM_NAMING_CONVENTION=$2
                LIST_BACKUPS=$(ls -tr "${BACKUP_DIR_PATH}")

  debugVar "LIST_BACKUPS" "${LIST_BACKUPS}"

                #default rotation if variable is not defined
                if [ -z ${VM_BACKUP_ROTATION_COUNT} ]; then
                                VM_BACKUP_ROTATION_COUNT=1
                fi
  debugVar                          "VM_BACKUP_ROTATION_COUNT"
"${VM_BACKUP_ROTATION_COUNT}"

                for DIR in ${LIST_BACKUPS};
                do
                  debugVar "DIR" "${DIR}"
                                TMP_DIR="${BACKUP_DIR_PATH}/${DIR}"
                  debugVar "TMP_DIR" "${TMP_DIR}"
                                TMP=$(expr  "${TMP_DIR}"  :  '.*\(--[0-9]*\)'  |
sed 's/-//g')
                                TMP=${TMP:-0}
```

```
                debugVar "TMP" "${TMP}"
                debugVar "BACKUP_VM_NAMING_CONVENTION"
"${BACKUP_VM_NAMING_CONVENTION}"
   if [ "${TMP}" = "${BACKUP_VM_NAMING_CONVENTION}" ]; then
     NEW="${TMP}--1"
     debugVar "Move newest" "$NEW"
     mv "${BACKUP_DIR_PATH}/${DIR}" "${NEW}"
   elif [ ${TMP} -ge ${VM_BACKUP_ROTATION_COUNT} ]; then
     debugVar "Remove old" "${BACKUP_DIR_PATH}/${DIR}"
     echo ">>>>>>>>>> Removing old backups..."
     rm -rf "${BACKUP_DIR_PATH}/${DIR}"
     echo ">>>>>>>>>> Done: Old backups removed"
   else
                                        BASE=$(echo   "${TMP_DIR%--
*}")
                NEW=${BASE}--$((${TMP}+1))
                debugVar "BASE" "$BASE"
                debugVar "NEW" "$NEW"
                debugVar       "BACKUP_DIR_PATH       +       DIR"
"${BACKUP_DIR_PATH}/${DIR}"
                  mv "${BACKUP_DIR_PATH}/${DIR}" "$NEW"
                fi
              done
              unset IFS
}
#---------------------------------------------------------
# Funkce : esxVmdBackup
#---------------------------------------------------------
esxVmdkBackup() {
  echo "This script has been tailored specifically for VMware Server 2"
  echo   "Please   see   lamw's   ESX(i)   ghettoVCB   script   at
http://communities.vmware.com/docs/DOC-8760"
}
#---------------------------------------------------------
# Funkce gsxVmdkBackup
#---------------------------------------------------------
gsxVmdkBackup() {
  IFS=":"
              echo   ">>>>>>>>>>   GSX:   Backing   up   content   of
${VM_NAME} ..."
  # Copy disks to backup location
  if [ ${RSYNC_FLAG} -eq 1 ]; then
    rsync          -av          --progress          --exclude=*-000001*.vmdk
${VMX_DIR}/*.vmdk "${VM_BACKUP_DIR}/"
  else
    cp                                  -p                                  -v
${VMX_DIR}/!(*000001*|*.log|*.vmx*|*.vms*|*.vmem|*.nvram|*.lck)
"${VM_BACKUP_DIR}/"
  fi
  # Take a copy of the VMX file to post-process
  VMX_BACKUP=${VM_BACKUP_DIR}/`basename ${VMX_PATH}`
  mv "${VMX_BACKUP}" "${VMX_BACKUP}.old"
              echo   ">>>>>>>>>>   GSX:   Reconfiguring   backup   of
${VM_NAME} ..."
  # Point the VMX to the original disk instead of the snapshot
  sed '/fileName/s/-000001.vmdk/.vmdk/g' < "${VMX_BACKUP}.old" >
"${VMX_BACKUP}"
  rm "${VMX_BACKUP}.old"
  unset IFS
  VM_COUNT_SUCCESS=$[ ${VM_COUNT_SUCCESS}+1 ]
}
#---------------------------------------------------------
# Funkce : archiveBackup
#---------------------------------------------------------
archiveBackup() {
  echo ">>>>>>>>>> Archiving the backup for ${VM_NAME}..."
  cd "${VM_BACKUP_DIR}"
  ${ARCHIVE_CMD} "${VM_NAME}.${ARCHIVE_EXT}" *
  echo ">>>>>>>>>> Removing files that have been archived..."
  find "${VM_BACKUP_DIR}" ! -name *.${ARCHIVE_EXT} -type f -
print0 | xargs -0 rm
}
#---------------------------------------------------------
# Funkce : terminate
#---------------------------------------------------------
terminate() {
  # Clean up temporary VM List
  if [ ${DEBUG_MODE} -eq 0 ]; then
    rm -f ${VM_LIST}
  fi
  if [ $1 -eq 1 ]; then
    echo -e "Error: Script did not complete."
    echo
"################################################
#####"
    exit 1
  else
    echo
"################################################
#####"
    exit 0
  fi
}
#---------------------------------------------------------
# Funkce : sanityCheck
#---------------------------------------------------------
sanityCheck() {
              NUM_OF_ARGS=$1

              if [ ! ${NUM_OF_ARGS} == 1 ]; then
                        printUsage
                        terminate 0
              fi

  VMWARE_CMD=`which vmware-vim-cmd`
  if [ ! -x $VMWARE_CMD ]; then
    if [ -f /usr/bin/vmware-vim-cmd ]; then
      VMWARE_CMD=/usr/bin/vmware-vim-cmd
                else
                  echo   "Cannot   locate   vmware-vim-cmd.   Are   you
running VMware Server?"
                  terminate 1
                fi
              fi
  if [ ! -z $VM_USER ]; then
    debugVar "VM_USER Detected" "$VM_USER"
    VMWARE_CMD="$VMWARE_CMD -U $VM_USER"
  fi
  if [ ! -z $VM_PASSWORD ]; then
    debugVar "VM_PASSWORD Detected" "$VM_PASSWORD"
    VMWARE_CMD="$VMWARE_CMD -P $VM_PASSWORD"
  fi

  RSYNC_FLAG=0
  if [ -x `which rsync` ]; then
    RSYNC_FLAG=1
  fi

  if [ -x `which bzip2` ]; then
    ARCHIVE_CMD="tar cjvf"
    ARCHIVE_EXT="tar.bz"
  elif [ -x `which gzip` ]; then
    ARCHIVE_CMD="tar czvf"
    ARCHIVE_EXT="tgz"
  else
    ARCHIVE_CMD="tar cvf"
    ARCHIVE_EXT="tar"
  fi

              if [ ! -f ${FILE_INPUT} ]; then
                        echo -e "Error: ${FILE_INPUT} is not a valid
VM input file!\n"
                        printUsage
              fi

  if [ ! -d ${VM_BACKUP_VOLUME} ]; then
```

```
   echo -e "Error: Backup location \"${VM_BACKUP_VOLUME}\" does not exist!\n"
   terminate 1
 elif [ ! -w ${VM_BACKUP_VOLUME} ]; then
   echo -e "Error: Backup location \"${VM_BACKUP_VOLUME}\" is not writable!\n"
   terminate 1
 fi

}
#-------------------------------------------------------------
# Funkce : serverVCB
#          SAMOTNA PROVADECI FUNKCE
#-------------------------------------------------------------
serverVCB() {

          VM_INPUT=$1

 START_TIME=`date`
 S_TIME=`date +%s`

 VM_LIST=`mktemp` || exit 1
          #dump out all virtual machines allowing for spaces
          ${VMWARE_CMD} vmsvc/getallvms | grep ^[0-9] | sed 's/[[:blank:]]\{3,\}/    /g' | awk -F'    ' '{print "\""$1"\";\""$2"\";\""$3"\""}' | sed 's/\] /\]\";\"/g' > $VM_LIST

          # Get the VMware Product Line
 getProduct
          debugVar "VMware Product Line" $productLineId

 OLD_IFS=$IFS
          IFS=$'\n'
          VM_COUNT=`cat "${VM_INPUT}" | sed '/^[[:blank:]]*$/d' | wc -l`
          VM_COUNT_SUCCESS=0
          VM_COUNT_FAIL=0
          VM_ITER=0
          for VM_NAME in `cat "${VM_INPUT}" | sed '/^[[:blank:]]*$/d'`;
   do
   VM_ITER=$[ ${VM_ITER}+1 ]
                    VM_ID=`grep -E "\"${VM_NAME}\"" ${VM_LIST} | awk -F ";" '{print $1}' | sed 's/"//g`

                    #ensure default value if one is not selected or variable is null
                    if [ -z ${VM_BACKUP_DIR_NAMING_CONVENTION} ]; then

          VM_BACKUP_DIR_NAMING_CONVENTION="$(date +%F)"
                    fi

                    VMFS_VOLUME=`grep -E "\"${VM_NAME}\"" ${VM_LIST} | awk -F ";" '{print $3}' | sed -e 's/\[//;s/\]//;s/"//g' -e 's/ /\\\ /g'`
                    VMX_CONF=`grep -E "\"${VM_NAME}\"" ${VM_LIST} | awk -F ";" '{print $4}' | sed 's/\[//;s/\]//;s/"//g'`
                    if [ -z "${VMFS_VOLUME}" ]; then
                      VMX_PATH="${VMX_CONF}"
                    else
                      getVimVar "hostsvc/datastore/info \"${VMFS_VOLUME}\"" path VMFS_PATH

VMX_PATH="${VMFS_PATH}/${VMX_CONF}"
                    fi

                    VMX_DIR=`dirname "${VMX_PATH}"`

                    debugVar "Virtual Machine" $VM_NAME
                    debugVar VM_ID $VM_ID
                    debugVar VMX_PATH $VMX_PATH
                    debugVar VMX_DIR $VMX_DIR
                    debugVar VMX_CONF $VMX_CONF
                    debugVar VMFS_VOLUME $VMFS_VOLUME
                    debugVar VM_BACKUP_DIR_NAMING_CONVENTION $VM_BACKUP_DIR_NAMING_CONVENTION

                    IFS=$OLD_IFS

                    #checks to see if we can pull out the VM_ID
                    if [ -z ${VM_ID} ]; then
                              echo "Error: Failed to extract VM_ID for VM identified as \"${VM_NAME}\"!"
                              echo ">!>!>!>!>! The following Virtual Machines are known:"
                              awk -F ";" '{print $2}' ${VM_LIST} | sed 's/"//g' | sort
                              echo "<!<!<!<!<! Ensure that name and case match!"
    VM_COUNT_FAIL=$[ ${VM_COUNT_FAIL}+1 ]

   #checks to see if the VM has any snapshots to start with
   elif ls "${VMX_DIR}" | grep -q -E "[0-9]{6}" > /dev/null 2>&1; then
              echo "Warning: Snapshot found for ${VM_NAME}, backup will not take place"
    VM_COUNT_FAIL=$[ ${VM_COUNT_FAIL}+1 ]
    continue

                    #checks to see if the VM has an RDM <- Not sure if Server can do this any more - better safe than sorry
                    elif ${VMWARE_CMD} vmsvc/device.getdevices ${VM_ID} | grep "RawDiskMapping" > /dev/null 2>&1; then
                              echo "Warning: RDM was found for ${VM_NAME}, backup will not take place"
    VM_COUNT_FAIL=$[ ${VM_COUNT_FAIL}+1 ]
    continue
   elif [[ -f "${VMX_PATH}" ]] && [[ ! -z "${VMX_PATH}" ]]; then
              BACKUP_DIR=`echo "${VM_BACKUP_VOLUME}/${VM_NAME}"`
              BACKUP_DIR_ESCAPED=`echo "${VM_BACKUP_VOLUME}/${VM_NAME}" | sed 's/ /\\\ /g'`
    if [[ -z ${VM_BACKUP_VOLUME} ]]; then
      echo "Error: Variable VM_BACKUP_VOLUME was not defined"
      terminate 1
    fi
                              #initial root VM backup directory
                              if [ ! -d "${BACKUP_DIR}" ]; then
                                        mkdir -p "${BACKUP_DIR}"
              fi
                              # directory name of the individual Virtual Machine backup followed by naming convention followed by count
                              VM_NAME_DIR=`echo ${VM_NAME} | sed 's/ /\\\ /g'`

          VM_BACKUP_DIR="${BACKUP_DIR}/${VM_NAME}-${VM_BACKUP_DIR_NAMING_CONVENTION}"
          VM_BACKUP_DIR_ESCAPED="${BACKUP_DIR_ESCAPED}/${VM_NAME_DIR}-${VM_BACKUP_DIR_NAMING_CONVENTION}"
                              debugVar VM_BACKUP_VOLUME "$VM_BACKUP_VOLUME"
                              debugVar BACKUP_DIR "$BACKUP_DIR"
                              debugVar BACKUP_DIR_ESCAPED "$BACKUP_DIR_ESCAPED"
                              debugVar VM_BACKUP_DIR "$VM_BACKUP_DIR"
                              debugVar VM_BACKUP_DIR_ESCAPED "$VM_BACKUP_DIR_ESCAPED"
```

```
    echo "++++++++++ Backup ${VM_ITER} of ${VM_COUNT}"
    echo "++++++++++ Initiating backup process for ${VM_NAME}"
                                        mkdir -p "${VM_BACKUP_DIR}"

                                        cp      -p      "${VMX_PATH}"
"${VM_BACKUP_DIR}"


            ORGINAL_VM_POWER_STATE=$(${VMWARE_CMD}
vmsvc/power.getstate ${VM_ID} | tail -1)
                        debugVar ORGINAL_VM_POWER_STATE
"$ORGINAL_VM_POWER_STATE"
                        debugVar
POWER_VM_DOWN_BEFORE_BACKUP
"$POWER_VM_DOWN_BEFORE_BACKUP"
                                        #section that will power down a
VM prior to taking a snapshot and backup and power it back on
                                        if                      [
${POWER_VM_DOWN_BEFORE_BACKUP} -eq 1 ]; then

            START_ITERATION=0
                                                echo "Powering off
initiated for ${VM_NAME}, backup will not begin until VM is off..."
                                                ${VMWARE_CMD}
vmsvc/power.shutdown ${VM_ID} > /dev/null 2>&1
                                                while
${VMWARE_CMD} vmsvc/power.getstate ${VM_ID} | grep -i
"Powered on" > /dev/null 2>&1;
                                                do

            START_ITERATION=$((START_ITERATION + 1))
                                                        #enable
hard power off code
                                                        if      [
${ENABLE_HARD_POWER_OFF} -eq 1 ]; then

            if          [           ${START_ITERATION}          -gt
${ITER_TO_WAIT_SHUTDOWN} ]; then

                        echo "Hard power off occured for
${VM_NAME}, waited for $((ITER_TO_WAIT_SHUTDOWN*3))
seconds"

                        ${VMWARE_CMD}           vmsvc/power.off
${VM_ID} > /dev/null 2>&1

                        #this is needed for ESXi, even the hard
power off did not take affect right away

                        sleep 5

                        break

            fi
                                                        fi
        echo "----------  VM ${VM_NAME} is still on -  Iteration:
${START_ITERATION} - waiting 3secs"
        sleep 3
                                                done
                                                echo ">>>>>>>>>>
VM ${VM_NAME} is off"
                                        fi

                                        #powered on VMs only
                                        if                  [[          !
${POWER_VM_DOWN_BEFORE_BACKUP}  -eq  1  ]]  &&  [[
"${ORGINAL_VM_POWER_STATE}" != "Powered off" ]]; then
                                                echo ">>>>>>>>>>
Taking backup snapshot for ${VM_NAME} ..."
                                                ${VMWARE_CMD}
vmsvc/snapshot.create               ${VM_ID}            vcb_snap
VCB_BACKUP_${VM_NAME}_`date +%F` > /dev/null 2>&1
                                                echo ">>>>>>>>>>
Done: Snapshot creation"
                                        fi
    if [ "${productLineId}" == "gsx" ]; then
      gsxVmdkBackup
      if [ ${ENABLE_ARCHIVING} -eq 1 ]; then
        archiveBackup
      fi
    else
      esxVmdkBackup
      terminate 0
    fi
                                        #powered      on      VMs      only
w/snapshots
                                        if                  [[          !
${POWER_VM_DOWN_BEFORE_BACKUP}  -eq  1  ]]  &&  [[
"${ORGINAL_VM_POWER_STATE}" == "Powered on" ]]; then
        echo ">>>>>>>>>> Removing snapshot from ${VM_NAME} ..."
                                                ${VMWARE_CMD}
vmsvc/snapshot.remove ${VM_ID} > /dev/null 2>&1
        getSnapshotTasks
        for TASK in ${SNAPSHOT_TASKS};
        do
          getSnapshotStatus "${TASK}"
        done
        echo ">>>>>>>>>> ${state}: Snapshot removal"

                                                #do      not      continue
until all snapshots have been committed
        echo ">>>>>>>>>> Waiting for snapshot on ${VM_NAME} to be
committed..."
                        while ls "${VMX_DIR}" | grep -q -E "[0-
9]{6}";
        do
          sleep 3
        done
        echo ">>>>>>>>>> Done: Snapshot committed"
                                                fi
                                                if                      [[
${POWER_VM_DOWN_BEFORE_BACKUP}  -eq  1  ]]  &&  [[
"${ORGINAL_VM_POWER_STATE}" == "Powered on" ]]; then
                                                #power on vm that
was powered off prior to backup
                                                echo ">>>>>>>>>>
Powering back on ${VM_NAME}"
                                                ${VMWARE_CMD}
vmsvc/power.on ${VM_ID} > /dev/null 2>&1
                                                fi
                                                checkVMBackupRotation
"${BACKUP_DIR}" "${VM_BACKUP_DIR}"
                                                echo -e "^^^^^^^^^^ Completed
backup for ${VM_NAME}!\n"
    else
      echo "Error: Failed to lookup ${VM_NAME}!"
      VM_COUNT_FAIL=$[ ${VM_COUNT_FAIL}+1 ]
    fi
  done
            unset IFS
            echo
  echo              "####################              Summary
############################"
  echo "Virtual Machines to backup: ${VM_COUNT}"
  echo "Successfully backed up    : ${VM_COUNT_SUCCESS}"
  echo "Failed to back up         : ${VM_COUNT_FAIL}"
  echo
"################################################################
#####"
  END_TIME=`date`
  E_TIME=`date +%s`
  echo "Start time: ${START_TIME}"
  echo "End   time: ${END_TIME}"
  DURATION=`echo $((E_TIME - S_TIME))`
  #calculate overall completion time
```

```
 if [ ${DURATION} -le 60 ]; then
        echo -e "Duration  : ${DURATION} Seconds\n"
 else
        echo  -e  "Duration      :  `awk  'BEGIN{  printf  "%.2f\n",
'${DURATION}'/60}'` Minutes\n"
 fi
                terminate 0
}
#------------------------------------------------------------
# ZACATEK SKRIPTU
#------------------------------------------------------------
echo
"#################################################
#####"
echo "# VMware Server Backup Script Version: $VERSION"
echo
"#################################################
#####"
# sanityCheck
sanityCheck $#
# hlavni program
serverVCB $1
```