FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY
UNIVERZITY KOMENSKÉHO V BRATISLAVE

Katedra informatiky

Práca procesorov v chránenom režime

Diplomová práca

Bratislava 2009 Peter Ambrož

Fakulta matematiky, fyziky a informatiky Univerzity Komenského v Bratislave
9.2.1 Informatika

Práca procesorov v chránenom režime

Diplomová práca

Peter Ambrož
Vedúci diplomovej práce: RNDr. Jaroslav Janáček
Bratislava 2009

Čestné prehlásenie

Čestne prehlasujem, že túto diplomovú prácu som vypracoval samostatne, len s použitím citovanej literatúry.

. . . . . .

Poďakovanie

Touto cestou by som sa chcel poďakovať môjmu školiteľovi RNDr. Jaroslavovi Janáčkovi, za podnetné nápady pri tvorbe tejto práce a za vysoko odbornú spoluprácu.

Abstrakt

Témou diplomovej práce je virtualizácia PC, spôsoby a využitie virtualizácie, fungovanie chráneného režimu v 32-bitových procesoroch, a rozdiely vo fungovaní na 64-bitových procesoroch rodiny Intel. Sústredíme sa najmä na techniky využité v operačných systémoch.
Dôležitou súčasťou práce je spustiteľný program, ktorý demonštruje vybrané procesy interaktívnym spôsobom. Cieľovou skupinou sú najmä začínajúci systémoví programátori.

Kľúčové slová: Chránený režim, virtualizácia, systémové programovanie

Abstract

Target of this diploma thesis is to describe virtualization techniques and usage, 32-bit protected mode and differences between 32-bit and 64-bit protected mode on the Intel architecture. We'll make emphasis on techniques used in operating systems.
Important part of thesis is executable program, which demonstrates selected processes in an interactive way. The main target is beginner or intermediate system programmer.

Key words: Protected mode, virtualization, system programming

Obsah

1 Úvod

Rozvoj počítačov ide míľovými krokmi vpred. Vyvíjajú sa nové technológie, či už po stránke hardwarovej alebo softwarovej. Virtualizácia je čoraz častejšie skloňované slovo v súvislosti s počítačmi a počítačovými technológiami. Nielen v oblasti komerčných poskytovateľov serverových riešení, ale aj medzi vývojármi softwaru a taktiež domácimi používateľmi je virtualizácia veľmi populárna.

Popíšeme si, čo znamená pojem "virtualizácia", aký má prínos v rôznych oblastiach informatiky. Taktiež sa pozrieme na rôzne prístupy ku virtualizácií, aké podmienky sú potrebné, aké problémy môžu vzniknúť, a ako sa s nimi vysporiadať. Samostatná kapitola je venovaná natívnej virtualizácii na platforme Intel x86, ktorá je v oblasti domácich používateľov a čiastočne aj serverových riešení dominantná.

Významným míľnikom vo vývoji procesorov je prechod na 64-bitovú technológiu. Budeme sa venovať zmenám oproti 32-bitovému režimu, novinkám a možnostiam behu 32-bitových aplikácií.

Ukážeme si nielen "učebnicový popis" fungovania procesora, ale aj autorove reálne skúsenosti pri tvorbe softwarovej časti, problémy, na ktoré sa dá naraziť a postupy, ktorými sa im dá vyhnúť.

Táto práca voľne naväzuje na dielo uvedené v [1], kde boli popísané základy problematiky chráneného režimu na procesoroch Intel i386 a vyšších. Niektoré okruhy chráneného režimu teda iba spomenieme pre úplnosť a odkážeme zvedavého čitateľa na vyššie spomínané dielo.

Priložený program umožní záujemcom odskúšať si niektoré aspekty správania sa procesora. Komentovaný zdrojový kód poslúži ako základ pre písanie ďalších aplikácií, napríklad ako základ operačného systému.

2 Adresácia

V tejto kapitole sa budeme zaoberať spôsobmi prekladu pamäťových adries. Hoci by sa zdalo, že najjednoduchšie by bolo priame použitie pamäťovej adresy tak, ako vystupuje na zbernici, nie je to tak. Pri malom množstve pamäte (povedzme do 64 kB) by to fungovalo bez problémov. Akonáhle však chceme pracovať s väčším množstvom adries, spúšťať viac programov naraz, a ešte k tomu každý program si pýta viacero nezávislých úsekov pamäte ku svojej činnosti, nezaobídeme sa bez istej abstrakcie nad pamäťovými adresami.

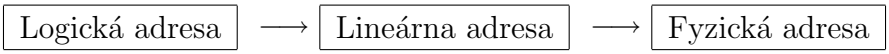
Pre lepšiu organizáciu miesta vo fyzickej pamäti procesor implementuje sadu pravidiel, podľa ktorých sa adresy použité v zdrojovom kóde prepočítavajú skôr, než sú odvysielané po fyzickej zbernici. Pri adresácií sa stretávame s troma typmi adries: Logická adresa, lineárna adresa, fyzická adresa. Typy adries majú nasledovný význam:

Logická adresa je adresa použitá v programovom kóde v nejakej inštrukcií, ktorá pracuje s pamäťou. Skladá sa z dvoch číselných hodnôt: Segment, offset. Segment sa pritom častokrát neuvádza, ale uvažuje sa implicitný segment podľa konkrétneho kontextu. Programátor a jeho programy pri práci s pamäťou využívajú práve logické adresy. Príklady logických adries: 0x10:231F, 0xF1A:711E4.

Lineárna adresa určuje pamäťové miesto v lineárnom adresnom priestore. Lineárny adresný priestor je predstava programátora, resp. jeho aplikácie o tom, ako vyzerá pamäť. Je to jediné čislo. V kontexte bežiacej aplikácie znamená poradové číslo pamäťovej bunky. V skutočnosti lineárna adresa nemusí korešpondovať s fyzickým rozložením pamäte. Toto sa však pred aplikáciami skrýva. Lineárny adresný priestor je vlastne virtuálna pamäť.

Fyzická adresa určuje miesto vo fyzickej pamäti, tak ako ju procesor vidí na zbernici počítača. Aplikácie s takýmito adresami vôbec nepracujú, a preklad lineárnej adresy na fyzickú je pred nimi skrytý. S fyzickými adresami pracuje iba systémový programátor, resp. jadro operačného systému, a aj to iba pri definovaní dôležitých štruktúr pre chod procesora.

Procesor používa 2 druhy prepočtu adries: Segmentáciu a stránkovanie. Segmentácia transformuje logické adresy na lineárne adresy. Stránkovanie má na starosti preklad lineárnych adries na fyzické adresy. Stránkovanie nemusí byť vždy aktívne. V prípade, že nie je aktívne, platí vzťah lineárna adresa = fyzická adresa.



Obr. 1: Postup pri adresácií.

2.1 Segmentácia

Lineárny adresný priestor sa z hľadiska aplikácií delí na úseky pamäte nazývané "segmenty". Segment má svoju lineárnu bázovú adresu, t.j. miesto, kde v pamäti začína, limit, ktorý určuje veľkosť segmentu, a atribúty.

Výpočet lineárnej adresy závisí od režimu, v ktorom sa procesor nachádza a konkrétne výpočty sa čitateľ dozvie napríklad v [1]. Všeobecne platí vzťah: Lineárna adresa = Báza + offset, za predpokladu, že offset $\leq$ limit. Báza a limit sa pritom získajú zo segmentu buď priamym výpočtom alebo nepriamo z tabuľky deskriptorov. V oboch prípadoch sa tento výpočet deje v procesore. V chránenom režime programátor môže zasahovať do tabuliek deskriptorov a upravovať bázy a limity segmentov, prípadne upraviť prístupové práva.

Problémy pri segmentácií ako napríklad neplatný segment, offset väčší ako limit, nedostatočné prístupové práva, vyvolávajú výnimku GP, general protection fault (všeobecná chyba ochrany). V špeciálnych prípadoch nastávajú aj výnimky SS, stack fault (chyba zásobníka) a NP, segment not present (výpadok segmentu).

Segmentácia uľahčuje alokáciu pamäte pre rôzne účely použitia. Môžeme mať samostatný segment pre kód, dáta, zásobník, prípadne samostatný segment pre niektoré privilegované funkcie OS. Na druhej strane to sťažuje správu pamäte z hľadiska programátora OS. Segmentácia sa dá čiastočne odstrániť použitím len dvoch segmentov - kódového a dátového - s bázou 0 a maximálnym limitom. Vtedy bude platiť vzťah offset = lineárna adresa.

2.2 Stránkovanie a virtuálna pamäť

Stránkovanie (angl. paging) umožňuje mapovať lineárne adresy na fyzické adresy. Ak je stránkovanie aktívne, procesor delí lineárny adresný priestor na úseky (stránky) fixnej veľkosti (4 kB, 2 MB, alebo 4 MB). Každá stránka môže byť mapovaná do fyzickej pamäte na určité miesto, prípadne označená ako neprítomná.

Ak stránka odkazovaná lineárnou adresou je označená ako neprítomná, dôjde ku výnimke PF, page fault (výpadok stránky). Obsluha výnimky v OS typicky pri takomto výpadku natiahne chýbajúcu stránku z disku do fyzickej pamäte, a prípadne odloží niektorú inú stránku na disk. Po tomto úkone sa vráti riadenie programu, ktorý vyvolal výnimku a pokus o prístup ku pamäti

je zopakovaný. Samotný program úspešne prečíta / zapíše dáta, a nedozvie sa, či pristupoval ku fyzickej pamäti alebo ku pevnému disku. Tejto technike sa hovorí "virtualizácia pamäte", alebo ľudovo povedané "swapovanie".

Operačný systém pomocou swapovania prezentuje aplikáciám veľký súvislý blok lineárnych pamäťových adries, pritom v skutočnosti môžu byť fyzické stránky umiestnené v pamäti aj v inom poradí a niektoré môžu byť neprítomné a obsluhované z iného média.

Informácia o mapovaní stránok je uložená v adresári stránok (page directory) a v tabuľkách stránok (page tables). Tieto štruktúry organizujú stránky lineárnej pamäte. O každej stránke existuje záznam v tabuľke stránok (page table entry, PTE), obsahujúci fyzickú bázovú adresu, prístupové práva a iné atribúty. Tabuliek stránok môže byť viac, a o každej z nich existuje záznam v adresári stránok (page directory entry, PDE), obsahujúci fyzickú bázovú adresu tabuľky, prístupové práva a iné atribúty. V najjednoduchšom prípade existuje jeden adresár stránok a jeho fyzická bázová adresa je umiestnená v registri CR3.

Mechanizmus stránkovania má rôzne rozšírenia. Podstatné vlajky, ktoré ovplyvňujú stránkovanie:

- CR0.PG, bit 31 v registri CR0, určuje, či je stránkovanie aktívne, alebo nie.
- CR4.PSE, povoľuje stránky o veľkosti 4 MB, resp. 2 MB.
- CR4.PAE, zapína adresáciu 36-bit fyzickej pamäte.

2.2.1 32-bit stránkovanie

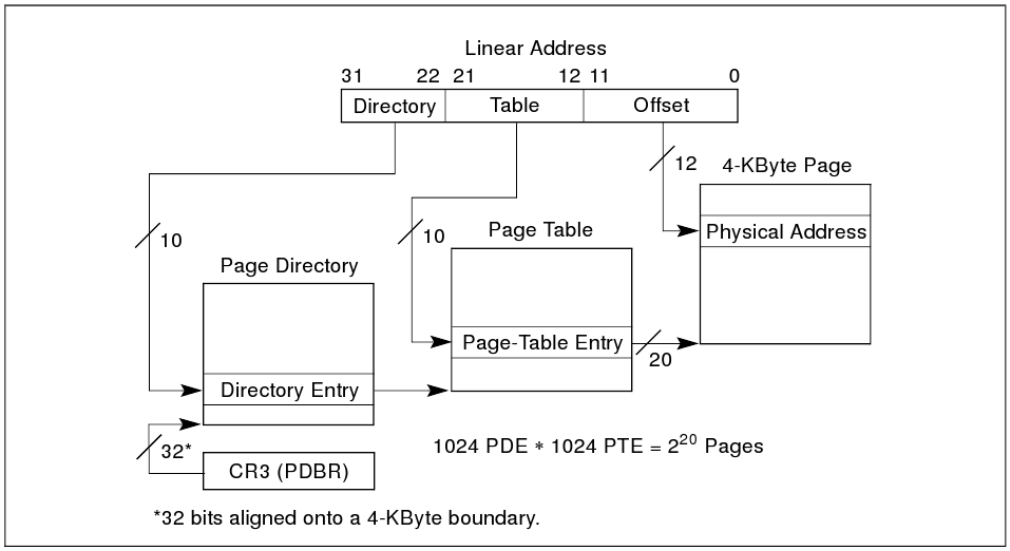
Stránkovanie prepočítava 32-bit lineárne adresy na 32-bit fyzické adresy. Týmto spôsobom je možné obslúžiť 2^{32} pamäťových adries, t.j. 4 GB fyzickej pamäte.

Procesor postupuje pri prepočte nasledovne: 32 bitov lineárnej adresy sa rozdelí na 3 polia. Prvých 10 bitov tvorí index č.1, druhých 10 bitov index č.2 a zvyšných 12 bitov tvorí offset. Index1 sa použije na výber záznamu z adresára stránok, je ním tabuľka stránok. Index2 sa použije na výber záznamu z tabuľky stránok, je ním konkrétna stránka vo fyzickej pamäti. Offset sa pripočíta ku báze vybratej stránky a vo výsledku dostávame 32-bit fyzickú adresu.

Tabuľky oboch úrovní obsahujú každá max. 1024 záznamov, offset určuje pozíciu v rámci 4 kB veľkej stránky. Situáciu najlepšie vystihne obrázok 2.

Alternatívne sa dajú použiť stránky s veľkosťou 4 MB. Treba k tomu zapnúť rozšírenie veľkosti stránok PSE (page size extension). Potom v ad-

resári stránok vieme pre jednotlivé záznamy selektívne zapnúť atribút PS, čo spôsobí, že záznam nebude ukazovať na tabuľku stránok, ale rovno na stránku. Lineárna adresa sa v tomto prípade podelí spôsobom 10 bitov index do adresára stránok, 22 bitov offset.



Obr. 2: Schéma 32-bit stránkovania, 4 kB stránky

2.2.2 36-bit stránkovanie pomocou PAE

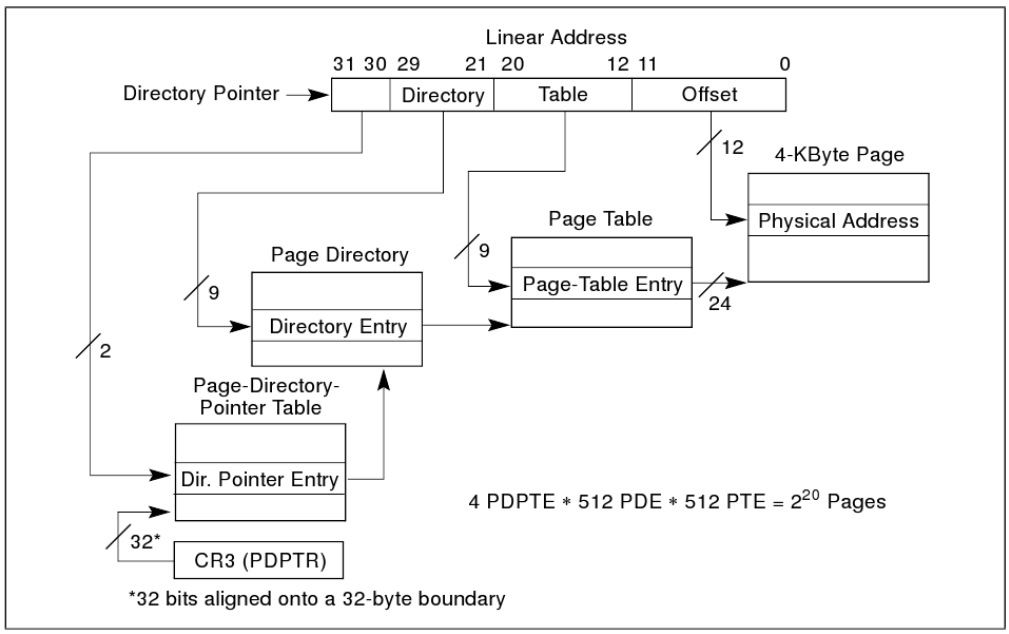
Pokiaľ potrebujeme obslúžiť viac ako 4 GB fyzickej pamäte, môžeme k tomu využiť technológiu PAE, physical address extensions, rozšírenie fyzických adries. Pomocou PAE sa dá prepočítavať 32-bit lineárne adresy na 36-bit fyzické adresy, dá sa teda sprístupniť až 64 GB pamäte v 4 GB veľkom okne lineárneho priestoru.

PAE zavádza tabuľku 3. úrovne s názvom "tabuľka ukazovateľov na adresáre stránok", anglicky "page directory pointer table", skrátene PDPT. Môže obsahovať až 4 záznamy, obsahujúce fyzickú adresu adresára stránok. Delenie lineárnej adresy vyzerá nasledovne: 2 bity pre index do PDPT, 9 bitov index do PD, 9 bitov index do PT, 12 bitov offset.

V tabuľkách 1. a 2. úrovne môže byť v každej max. 512 záznamov. Všetky záznamy obsahujú 36-bit fyzickú adresu, zarovnanú na 4 kB (t.j. ukladá sa horných 24 bitov, zvyšné sa implicitne považujú za nulové).

Postupuje sa podobne ako pri 32-bit stránkovaní, indexy vyberajú jednotlivé záznamy v tabuľkách, až sa dopracuje ku fyzickej adrese vybranej

stránky. K tejto sa pripočíta offset a získa sa tak 36-bit fyzická adresa, ktorú procesor použije. Viď obr. 3.



Obr. 3: Schéma 36-bit stránkovania pomocou PAE, 4 kB stránky

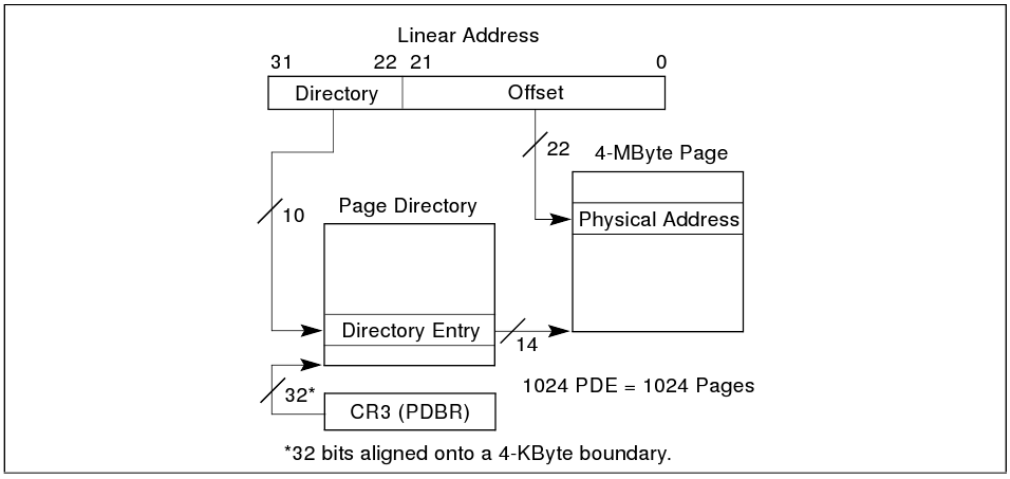
Aj pri PAE máme možnosť použiť veľké stránky, v tomto prípade 2 MB. Bit PS v PDE zázname rozhodne, či záznam ukazuje na tabuľku stránok, alebo na stránku o veľkosti 2 MB. Offset má 21 bitov.

2.2.3 36-bit stránkovanie pomocou PSE-36

Alternatívou ku PAE je technológia PSE-36. Aj táto umožňuje adresovať 36-bit fyzické adresy. Treba nastaviť bit PSE, a vypnúť bit PAE. Rozdelenie lineárnej adresy je v tomto prípade 10 bitov index do adresára stránok, 22 bitov offset. Záznamy v adresári stránok musia mať nastavený bit PS, a bázové adresy majú šírku 14-bit. Záznamy ukazujú priamo na stránky o veľkosti 4 MB. Po doplnení implicitných 22 núl a pripočítaní offsetu dostávame 36-fyzickú adresu.

Nevýhoda oproti PAE spočíva v nemožnosti použiť menšie, 4 kB stránky. Výhodou je jednoduchšie spravovanie, keďže sa používa len adresár stránok, pritom PAE vyžaduje až 3 tabuľky v hierarchií.

Obrázok 4 ukazuje stránkovanie pomocou PSE-36.



Obr. 4: Schéma 36-bit stránkovania pomocou PSE-36, 4 MB stránky

2.2.4 Prehľad stránkovacích režimov

Tabuľka č.5 ukazuje možné kombinácie vlajok a výsledný efekt na stránkovanie. Symbol X v políčku znamená, že daný bit nie je podstatný.

PG	PAE	PSE	PS (PDE)	PSE-36	Veľkosť stránky	Fyzická adresa
0	X	X	X	X	Stránkovanie vypnuté	
1	0	0	X	X	4 kB	32-bit
1	0	1	0	X	4 kB	32-bit
1	0	1	1	0	4 MB	32-bit
1	0	1	1	1	4 MB	36-bit
1	1	X	0	X	4 kB	36-bit
1	1	X	1	X	2 MB	36-bit

Obr. 5: Prehľad možností stránkovania

3 Režimy práce

Režim práce[1] (alebo tiež mód) je stav, v ktorom sa procesor môže nachádzať. V pôvodnom 16-bit procesore 8086 existoval iba jediný mód, ktorý sa dodnes zachoval pod názvom "reálny mód". Tento postačoval svojimi vlastnosťami pre beh jednoduchých aplikácií, ktoré nevyžadujú veľké množstvo pamäte. Postupom času tento režim už nestačil, a z hľadiska kompatibility s existujúcimi aplikáciami, ostal tento režim zachovaný a pribudol ďalší. Tento trend sa zachováva v procesoroch Intel dodnes, a pri zásadnej zmene koncepcie sa vytvorí nový režim, pričom staršie sa zachovajú pre kompatibilitu. Každý si tak môže vybrať, do akej miery chce procesor využiť, a podľa toho si vyberie aj režim, v ktorom bude fungovať.

V každom režime je presne definované správanie procesora pri spracovávaní inštrukcií. Rozdiely sú najmä v množine povolených inštrukcií, v spôsobe adresácie, v prístupe ku I/O zariadeniam, v spracovaní prerušení a v prostredí.

Rozoznávame 3 režimy: **Reálny**, **Chránený**, **Virtual-8086**. V pentiu a vyšších existuje aj 4. režim: **SMM**. Podrobnejší popis týchto režimov je možné nájsť v [1], nasleduje len stručný popis. S príchodom 64-bitovej architektúry prichádzajú ďalšie 2 módy: **IA-32e Kompatibilný** a **IA-32e 64-bit**.

3.1 Reálny mód

V tomto móde sa nachádza procesor vždy po zapnutí alebo po resete. Prostredie je 16-bitové, adresácia je reálna, systém ochrany je vypnutý a je možné využívať niektoré privilegované inštrukcie. Stránkovanie je v reálnom móde vypnuté, takže lineárna adresa = fyzická adresa. Prístup ku I/O zariadeniam je neobmedzený.

3.2 Chránený mód

Chránený mód je natívny mód, v ktorom je možné využiť naplno všetku funkcionalitu procesora a adresovať celú dostupnú pamäť. Prostredie je 32-bitové, adresácia je chránená, taktiež je zapnutý systém ochrany. Nie všetko je dovolené - týka sa to najmä privilegovaných inštrukcií, prístupu do pamäte a prístupu ku I/O zariadeniam. Je podporovaný multitasking. Dá sa zapnúť aj stránkovanie a využiť tak virtualizáciu pamäte.

[1] angl. Operation mode

3.3 Virtual-8086 mód (V86)

Tento mód je určený pre beh procesov písaných pre reálny mód pod chráneným módom. Je tu reálna adresácia so zapnutým stránkovaním, 16-bitové prostredie a celé sa to tvári ako ozajstný reálny mód. Toto všetko sa však deje vo vnútri chráneného módu a každý "prehrešok" V86 procesu voči systému ochrany vyvoláva výnimku. Obsluha výnimiek pre V86 mód môže emulovať správanie reálneho módu a vrátiť riadenie procesu, ktorý sa o výnimke nedozvie.

3.4 SMM mód

System Management Mode - mód na údržbu systému sa používa na beh systémových utilít s plnými právami. Prostredie je 32-bitové, adresácia je podobná reálnej adresácií, ale offset má 32-bitov, takže možno obsiahnuť celú dostupnú pamäť. Systém ochrany dovolí všetko. Prepnutie do SMM sa dá vykonať iba hardwarovo - privedením impulzu na SMI pin procesora alebo pomocou prerušenia od systému APIC. Do SMM režimu sa procesor prepne z ľubovoľného iného režimu plne automaticky a bez toho, aby sa narušil stav aktuálne vykonávanej úlohy. SMM režim sa ukončuje inštrukciou RSM, ktorá navráti procesor do pôvodného stavu, v ktorom bol pred prerušením.

3.5 64-bitové módy

Týmto režimom sa podrobnejšie venuje kapitola 4.

3.5.1 IA-32e kompatibilný

Prostredie pre spúšťanie programov písaných pre 32-bitový chránený režim. Prostredie je 32-bitové, väčšina programov pobeží bezo zmeny. Výhoda oproti použitiu 32-bit chráneného režimu je v možnosti využiť 64-bit registre a 64-bit lineárnu adresáciu. Výhodou je tiež to, že procesor beží v IA-32e režime a ostatné procesy v systéme tak môžu byť vykonávané v natívnom režime. Prepínanie medzi kompatibilným a natívnym režimom sa deje pomocou jediného bitu v registri kódového segmentu.

3.5.2 IA-32e 64-bit (natívny)

Prostredie pre 64-bitové programy. Štandartná veľkosť operandov je 32-bit, ale dá sa pretypovať na 64-bit. Adresácia je lineárna, 64-bit. Programy písané pre 32-bit nebudú bežať v tomto móde, lebo došlo ku viacerým zmenám v správaní sa systému ochrany.

4 Intel 64 architektúra

Na poli desktopových procesorov dlhú dobu kraľovali 32-bit procesory s natívnym 32-bit režimom práce. Postupom času sa však ukázalo, že 32 bitov spôsobuje obmedzenia vo viacerých smeroch. Jednak sme obmedzení na aritmetické výpočty s číslami o veľkosti 32-bit v rámci 1 operácie. Výrazným problémom je aj veľkosť (a teda aj počet) pamäťových adries. Limit 4 GB pamäte v jednom počítači je už dlhšiu dobu prekonaný. Existujú síce viacmenej úspešné spôsoby, ako pracovať aj s väčším množstvom pamäte (napr. PAE technológia rozširuje fyzické adresy na 36-bit, čo je 64 GB pamäte). Ale aj takéto rozšírenia neriešia problém úplne, pretože lineárne adresy sú pevne dané, 32 bitové. Aplikácia tak naraz nevidí viac ako 4 GB pamäte.

Situáciu rieši až príchod novšej architektúry procesorov s novým režimom práce. Intel 64 architektúra rozširuje lineárny adresný priestor na 64 bitov a podporuje adresovanie 40-bitov fyzickej pamäte. Pre využitie tejto technológie pribudli v procesoroch režimy IA-32e. V 64-bit režimoch môžu aplikácie využiť nasledovné novinky:

- 64-bit lineárne adresovanie bez segmentácie (tiež "flat")
- 8 prídavných registrov pre všeobecné použitie
- 8 prídavných registrov pre inštrukcie SSE, SSE2, SSE3
- 64-bit šírka všeobecných registrov aj ukazovateľa inštrukcií
- systém pre prioritizáciu prerušení
- nový spôsob adresácie - relatívny ku ukazovateľu inštrukcií
- zmeny v systéme ochrany za účelom zjednodušenia fungovania

Procesory Intel 64 architektúry aj naďalej podporujú všetky predošlé režimy práce, takže existujúci software (či už systémový alebo aplikačný) nie je treba upravovať. Pri použití IA-32e kompatibilného režimu je dokonca možné využiť niektoré výhody 64-bit architektúry pri zachovaní kompatibility s pôvodným systémom ochrany.

4.1 Špecifické registre MSR

V ďalšom texte sa bude často vyskytovať pojem MSR - model specific register. MSR sami o sebe nie sú novinkou v Intel 64 architektúre, avšak práve tu sa začali hojnejšie využívať. Jedná sa o registre procesora, do ktorých sa ukladajú špeciálne konfiguračné hodnoty, ktoré majú význam častokrát len v

niektorých verziách procesora, a ovplyvňujú správanie niektorých inštrukcií. MSR sa dajú prirovnať ku konfiguračným registrom CRx, avšak registre CRx plnia všeobecnejšiu funkciu, nastavuje sa v nich globálne správanie procesora.

Prístup ku registrom MSR je zabezpečený pomocou inštrukcií RDMSR (čítanie) a WRMSR (zápis). V oboch prípadoch sa použije parameter - číslo registra MSR. Vyčerpávajúci zoznam dostupných MSR sa dá nájsť v príručke od Intelu, v diele [6]. Inštrukcie pre prácu s MSR sú privilegované, vyžadujú úroveň CPL=0.

Medzi zaujímavé MSR patria:

TSC - Time stamp counter, počítadlo tikov procesora od posledného resetu. Dá sa použiť ako pomerne presný zdroj hodinových tikov, ale v praxi sa volia skôr iné metódy na presné sledovanie času, najmä z dôvodu prenositeľnosti na iné architektúry.

PMC - Performance monitor counter, 40-bit počítadlo udalostí v procesore. Dá sa naprogramovať, aké udalosti sa majú sledovať. Môže to byť počet dekódovaných inštrukcií, počet skokov, počet vonkajších prerušení, ...

IA32_EFER - Obsahuje príznaky (flags) týkajúce sa IA-32e režimov, viď časť 4.8.

4.2 Systémové štruktúry

Systémové registre GDTR, LDTR, IDTR majú rozšírenú veľkosť bázovej adresy na 64 bitov. Podobne je to v prípade kontrolných registrov CR0-CR4, debug registrov DR0-DR7, ktorých veľkosť vzrástla na 64 bitov. Všeobecné registre EAX, EBX, ECX, EDX, EDI, ESI, EBP, ESP, stavový register EFLAGS a tiež ukazovateľ inštrukcie EIP sa rozšírili na 64 bitov, nové registre sú dostupné pod názvami začínajúcimi písmenom R, teda RAX, RBX, RFLAGS, RIP, ... Spodných 32 bitov týchto registrov ostávajú dostupné pod pôvodnými názvami. Zachovávajú sa tiež 16 bitové registre (AX, BX, ...) a 8 bitové (AH, AL, BH, BL, ...). Novinkou sú registre R8-R15 pre všeobecné použitie.

Tabuľky GDT, LDT, IDT môžu obsahovať deskriptory o veľkosti 8 B pre kódové a dátové segmenty, a tiež 16 B pre volacie brány, TSS, LDT a brány prerušení (interrupt gate, trap gate). Nová veľkosť 16 B pre systémové deskriptory umožňuje uložiť 64 bitovú lineárnu bázovú adresu.

Miešanie deskriptorov rôznych veľkostí v jednej tabuľke deskriptorov nespôsobuje nejednoznačnosti pri ich interpretácií. Nové 16 B deskriptory dodržujú vhodný formát. Obidve časti, horných 8 bajtov aj spodných 8 bajtov,

vyzerajú ako samostatné 8 B deskriptory, pričom spodná časť má v typovom poli uložený skutočný typ deskriptora. Horná časť má nastavený typ=0, ktorý bol v doterajších procesoroch rezervovaný a aj naďalej ostáva rezervovaný. Natívny 64-bit systém vie, pri ktorých deskriptoroch má očakávať a rozpoznávať dáta aj v horných 8 bajtoch. Kompatibilný režim zasa vidí rozšírené časti deskriptorov ako neplatné deskriptory rezervovaného typu a ignoruje ich. Selektory (indexy do tabuliek deskriptorov) naďalej počítajú s veľkosťou 8 B na jeden záznam. Preto v prípade použitia 16 B deskriptorov sa používa iba každý druhý index. 16 B deskriptor jednoducho zaberá miesto pre dva 8 B deskriptory. Pri indexovaní sa použije ten selektor, ktorý ukazuje na spodnú časť 16 B deskriptora.

Veľkosť operandov väčšiny bežných inštrukcií ostáva naďalej 32-bit. Pre využitie 64-bit parametrov treba použiť pred každou inštrukciou prefix 0x48, nazývaný tiež REX.W, ktorý zabezpečí pretypovanie veľkosti operandu.

4.3 Segmentácia

V režime IA-32e kompatibilnom funguje segmentácia a súvisiaci systém ochrany tak ako v pôvodnom chránenom režime. V prípade IA-32e 64-bit došlo ku viacerým zmenám. Segmentácia je takmer úplne vypnutá. Vytvára sa tak jednoliaty 64 bitový lineárny adresný priestor. Pre uplatňovanie ochrany častí pamäte naďalej slúži systém stránkovania.

Segmentové registre CS, DS, ES, SS automaticky uvažujú lineárnu bázovú adresu 0 a procesor ignoruje informáciu v deskriptore. Segmenty FS, GS sa považujú za systémové a pri ich použití sa berie do úvahy aj bázová adresa z deskriptora. Limity segmentov sa nekontrolujú vôbec a to ani v prípade segmentov FS, GS. Namiesto toho sa kontroluje, či je výsledná lineárna adresa v kanonickom tvare.

Požiadavka na kanonický tvar hovorí, aby sa nepoužívali adresy vyššie ako tie, ktoré sa dajú skutočne použiť. Aktuálna architektúra Intel 64 implementuje preklad 48-bit lineárnych adries na fyzické adresy. Z dôvodu zachovania obojsmernej kompatibility sa požaduje, aby programátori nepoužívali adresy mimo tohto rozsahu. Znamená to teda, že bity 48-63 lineárnej adresy musia ostať všetky nulové (poprípade všetky jednotkové).

Segment CS si zachováva informáciu o CPL, a taktiež je využité pole DPL z deskriptora. Tieto informácie sa naďalej využívajú v systéme ochrany.

V deskriptore kódového segmentu pribudol bit L (53. bit), ktorý rozlišuje medzi kompatibilným a natívnym režimom vykonávania inštrukcií. Pomocou tohto bitu sa dá pre každý kódový segment zvlášť určiť, či sa jeho kód bude vykonávať podľa pravidiel kompatibilného režimu (L=0), alebo podľa pravidiel 64-bit režimu (L=1). Všimnime si, že v prípade L=0 budú platiť aj

bázové adresy, bude fungovať kontrola limitov, adresy budú 32 bitové.

Všetky segmenty môžu byť naplnené nulovým selektorom. Procesor túto akciu povolí aj pri plnení segmentov CS, SS. Použitie (dereferencia) nulového segmentu je tiež umožnené a nevyvoláva výnimku.

Deskriptor volacej brány (call gate) neobsahuje pole s počtom parametrov, a ukladá offset cieľového segmentu v šírke 64-bit. Volacia brána musí ukazovať na 64-bit kódový segment (L=1). Pri použití brány sa na zásobník ukladajú 64-bit hodnoty (aj segmentové registre sa doplnia nulami na 64-bit). Ku kopírovaniu parametrov nedochádza. Toto si musí zabezpečiť volajúca resp. volaná funkcia a to aj v prípade, že došlo ku zmene CPL. V prípade zmeny CPL dochádza ku zmene zásobníka tak, ako to bolo v 32-bit chránenom režime. Segment SS sa naplní hodnotou null (0) a pole RPL sa nastaví na hodnotu nového CPL. Register RSP (64-bit verzia ESP, ukazovateľ zásobníka) sa naplní hodnotou z TSS pre novú úroveň CPL. Staré hodnoty SS, RSP sú uložené na nový zásobník.

4.4 Stránkovanie

Systém stránkovania je rozšírený o tabuľku stránok 4. úrovne. Zároveň je aktívne rozšírenie PAE, ktoré pridáva tabuľku 3. úrovne. Všetky 4 tabuľky sú indexované 9 bitmi z lineárnej adresy a spolu s 12 bitmi pre určenie offsetu v stránke toto dáva dokopy 48 bitov lineárneho priestoru. Toto sa týka použitia stránok o veľkosti 4 kB.

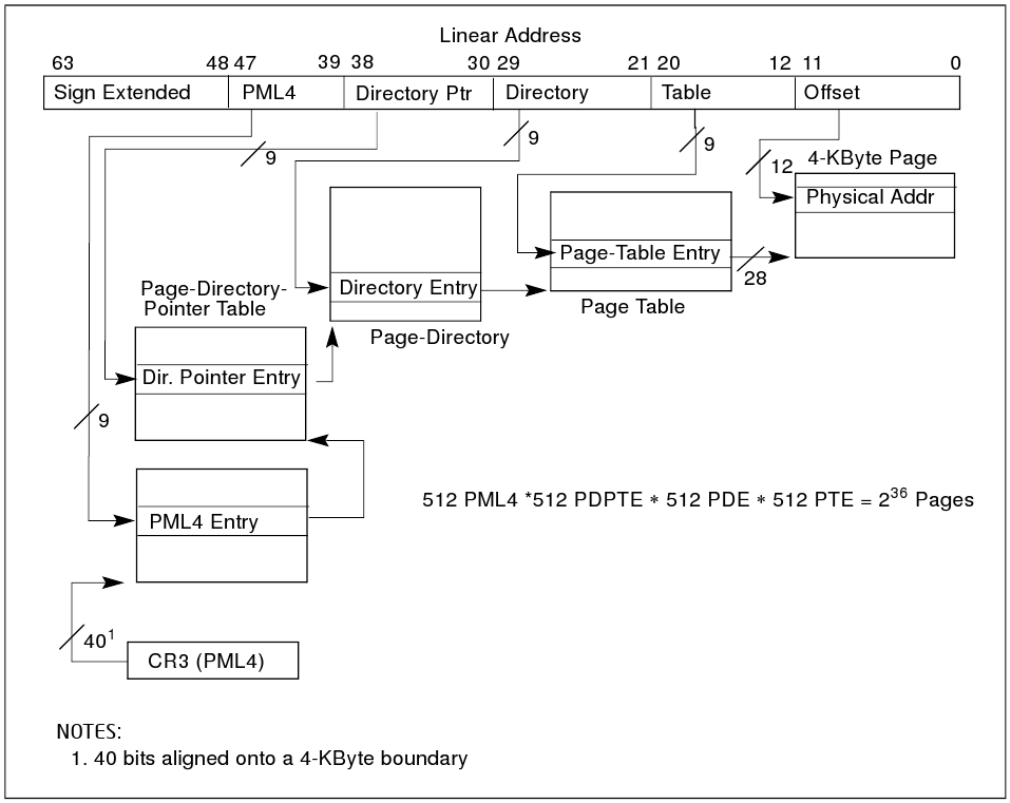
Ak v tabuľke adresára stránok (tabuľka druhej úrovne) zapneme bit PS, môžeme využívať stránky o veľkosti 2 MB. V takom prípade bude delenie lineárnej adresy nasledovné: 9-bit indexy do tabuliek 4., 3. a 2. úrovne, a 21-bit offset v rámci stránky.

Viac ako 48 bitov sa zatiaľ využiť nedá a vyššie bity sú rezervované do budúcnosti. Výsledkom stránkovania je 40 bitová fyzická adresa. Teoreticky je tento systém schopný prekladať 64-bit lineárne adresy na 52-bit fyzické adresy.

4.5 Prepínanie úloh

Prepínanie úloh v 64-bit režime nie je k dispozícií. Z hľadiska HW sú všetky inštrukcie vykonávané v rámci jedinej úlohy a každý pokus o prepnutie úlohy (Task switch) pomocou CALL alebo JMP na deskriptor úlohy (TSS) vyvolá výnimku (GP). Multitasking treba riešiť softwarovo.

Štruktúra stavu úlohy (Task State Segment, TSS) však naďalej existuje a register TR (Task Register) by mal ukazovať na platný deskriptor TSS. TSS už nie je odkladací priestor pre stav úlohy, ale obsahuje 64-bit ukazovatele



Obr. 6: Schéma stránkovania v režime IA-32e

preddefinovaných zásobníkov IST1 - IST7, ukazovatele zásobníkov pre zmenu CPL: RSP0 - RSP2, a tiež I/O mapu.

4.6 Obsluha prerušení

Obsluha všetkých prerušení (či už sú to výnimky, externé prerušenia alebo softwarovo vyvolané prerušenia) musí byť 64 bitová. Všetky hodnoty ukladané procesorom na zásobník sú doplnené a zarovnané na 64 bitov. Dvojica SS:RSP je uložená na zásobník pred každým volaním obslužnej rutiny a to bez výnimky. V 32-bit chránenom režime sa ukazovatele zásobníka ukladali len pri zmene CPL. Nepodmienené ukladanie stavu zásobníka uľahčuje a zjednocuje obsluhu. Pri zmene CPL dochádza ku výmene zásobníka tak, ako sa to deje aj pri volaní cez bránu.

Deskriptory brán prerušení (interrupt, trap) zaberajú 16 B a okrem bežných polí obsahujú 64-bit offset obslužnej rutiny. V deskriptore sa nachádza tiež nové pole IST (Interrupt Stack Table), so šírkou 3 bity.

Pole IST slúži na výber vopred definovaného zásobníka pre obsluhu daného prerušenia. Toto sa hodí pri obsluhe výnimiek, pri ktorých môže byť pôvodný zásobník poškodený a nevhodný na použitie v obslužnej rutine. Je možné zvoliť si jeden zo 7 preddefinovaných zásobníkov 1-7 (konkrétne hodnôt registra RSP), pritom tieto sú definované v TSS segmente. Hodnota 0 znamená, že chceme vymeniť zásobník podľa starých pravidiel: Ak nedošlo ku zmene CPL, nebude sa meniť. Ak sa zmenilo CPL, nový zásobník sa získa z TSS podľa nového CPL.

4.7 Ostatné

Dve nové inštrukcie SYSCALL a SYSRET slúžia na rýchle systémové volania z CPL 3 do CPL 0, resp. návrat z CPL 0 do CPL 3. Funkcie nemajú žiaden parameter. Cieľový segment a offset, a tiež segment zásobníka sú uložené v registroch MSR. Zároveň sa pri volaní aplikuje maska na stavový register RFLAGS. Táto maska sa opäť ukladá do príslušného MSR registra. Volania cez inštrukcie SYSCALL / SYSRET sú rýchle, lebo využívajú prednastavené hodnoty, ktoré už netreba kopírovať v pamäti a kontrolovať systémom ochrany.

Pri použití SYSCALL sa efektívne udejú nasledovné kroky:

- Kódový segment CS sa naplní selektorom z MSR IA32_STAR (bity 32-47), CPL sa nastaví na 0, DPL=0, báza=0 a limit maximálny.
- Ukazovateľ inštrukcie RIP sa naplní z MSR IA32_LSTAR a pôvodný RIP sa skopíruje do registra RCX.
- Stavový register EFLAGS sa bitovo maskuje (operácia AND) s MSR IA32_FMASK a pôvodná hodnota sa uloží do R11.
- Segment zásobníka SS bude obsahovať selektor CS + 8.

Pri návrate pomocou SYSRET sa CPL nastaví na 3, EFLAGS a RIP sa obnovia z registrov R11, resp. RCX. Segment SS nadobudne hodnotu MSR IA32_STAR + 8 (bity 48-63), a segment CS sa obnoví z MSR IA32_STAR (48-63) v prípade návratu z 64-bit režimu. Pri návrate z kompatibilného režimu sa použije IA32_STAR + 16 (48-63).

Vidíme, že ukazovateľ zásobníka RSP sa vôbec neupravuje. Počíta sa s tým, že volaná funkcia nepotrebuje oddelený zásobník, a všetko si po sebe riadne uprace, alebo vykoná výmenu zásobníka samotná funkcia, ak to potrebuje. Toto opäť nepatrne, ale predsa šetrí réžiu volania, čo bolo od začiatku cieľom týchto inštrukcií.

4.8 Prepnutie do IA-32e režimu

Vstup do IA-32e režimu je možný z 32-bit chráneného režimu, a ukončenie IA-32e znamená návrat do 32-bit chráneného režimu. Prepnutie do Virtual-8086 režimu z IA-32e nie je možné (generuje výnimku GP).

Nasledujúce kroky popisujú prepnutie do IA-32e:

- Vypnúť stránkovanie zápisom hodnoty 0 do CR0.PG.
- Zapnúť rozširenie stránkovania PAE zápisom do CR4.PAE. Bez rozšírenia PAE nebude IA-32e režim fungovať.
- Uložiť bázovú adresu tabuľky stránok 4. úrovne do registra CR3.
- Zapnúť režim IA-32e zápisom do MSR registra IA32_EFER.LME=1.
- Povoliť stránkovanie zápisom CR0.PG=1. Procesor následne nastaví hodnotu IA32_EFER.LMA=1, čím indikuje aktívny IA-32e režim. Aby sa zmena v stránkovaní prejavila ihneď, je vhodné použiť inštrukciu skoku, čím sa vyprázdni fronta predspracovaných inštrukcií.

Po prepnutí si niektoré systémové registre uchovávajú starú, 32-bit bázovú adresu, keďže boli plnené v 32-bit chránenom režime. Týka sa to registrov CR3, GDTR, LDTR, IDTR, TR. Štruktúry ako tabuľka stránok, alebo tabuľky deskriptorov sa teda zatiaľ nachádzajú v pamäti pod hranicou 4 GB. Nič nám nebráni teraz nanovo naplniť uvedené registre 64-bit hodnotami a prípadne tak presunúť niektoré systémové tabuľky do vyššej pamäte (nad 4 GB).

Tabuľku pre obsluhu prerušení (IDT) treba čím skôr zameniť za novú, ktorá obsahuje 16 B deskriptory. Kým sa tak stane, treba sa vyhýbať prerušeniam. Externé prerušenia sa dajú zakázať inštrukciou CLI. Nemaskovateľné prerušenie NMI je treba zakázať v externom obvode.

Voľba medzi kompatibilným a natívnym (64-bit) režimom sa deje nastavením bitu L v kódovom segmente. Pre každý kódový segment takto vieme určiť režim, v ktorom sa má jeho kód vykonávať.

Opustenie IA-32e režimu vyžaduje nasledovné kroky:

- Prepnúť sa do IA-32e kompatibilného režimu (skok do segmentu s L=0).
- Deaktivovať stránkovanie zápisom CR0.PG=0. Tým sa deaktivuje aj IA-32e režim a procesor zapíše do MSR registra IA32_EFER.LMA=0.
- Voliteľne vypnúť PAE pomocou CR4.PAE=0.

- Zapísať do CR3 novú 32-bit bázovú adresu tabuľky stránok 3. resp. 2. úrovne.
- Vypnúť IA-32e režim zápisom do MSR IA32_EFER.LME=0.
- Voliteľne zapnúť stránkovanie pomocou CR0.PG=1 a následne použiť skok, aby sa vyčistila fronta predspracovaných inštrukcií.

5 Virtualizácia

Zjednodušene povedané, virtualizácia znamená delenie výpočtovej sily jedného fyzického stroja medzi viacero systémov, bežiacich nezávisle a úplne oddelene od seba. Virtualizácia sa stala veľmi dôležitou technológiou v momente, keď výkon počítačov stúpol natoľko, že pri bežnom nasadení nie je naplno využitý. Keďže výkon naďalej stúpa a moorov zákon stále funguje, virtualizácia je jedna z ciest ku plnému využitiu zdrojov.

V počítačovej terminológií je virtualizácia široký pojem označujúci abstrakciu nad zdrojmi[2]. Tiež je to skrývanie fyzických vlastností výpočetných zdrojov alebo samotných zdrojov pred aplikáciami, ktoré s nimi pracujú. Virtualizácia zahŕňa predstieranie, že jeden fyzický zdroj (pevný disk, celý server, alebo OS) navonok vystupuje ako viacero zdrojov. Je možné virtualizovať aj opačným smerom - t.j. viacero zdrojov navonok vystupuje ako jediný. Taktiež jedno zariadenie (zdroj) môže vystupovať ako to isté zariadenie s inými vlastnosťami.

Technológia virtualizácie siaha v histórií až do 60-tych rokov 20. storočia. Vždy to znamenalo spôsob skrývania technických detailov pomocou enkapsulácie. Virtualizácia vytvára vonkajšie rozhranie, ktoré zakrýva vnútornú implementáciu. Pri prístupe ku fyzickému zariadeniu používa multiplex (pokiaľ ide o virtualizáciu viacerých zariadení nad jedným fyzickým), prípadne kombinuje viacero fyzických zariadení.

Dôvodov pre virtualizáciu je mnoho a sú viaceré cieľové skupiny, ktoré ju využívajú:

- **Domáci používateľ** využije virtualizáciu ku spúšťaniu iného operačného systému vovnútri svojho existujúceho OS bez nutnosti ukončiť všetku prácu a resetu počítača. Môže tak používať napr. Linux, a pokiaľ potrebuje použiť aplikáciu písanú pre Windows, jednoducho si spustí Windows ako aplikáciu v Linuxe (virtualizácia celého PC). Tatkiež môže využiť aplikačnú virtualizáciu a spustiť Win32 aplikáciu priamo pod Linuxom (opäť s využitím vhodného programu na virtualizovanie OS prostredia).

- **Systémový programátor** vyvíja ovládače zariadení alebo kompletný OS. Aby nemusel svoju robotu neustále prerušovať testovaním na inom PC, použije prostredie virtuálneho PC v ktorom môže testovať výsledky svojej práce. Výhodou je tiež izolácia, takže ak by došlo ku chybe v programe, nemôže prísť o dáta na svojom existujúcom PC. Najhoršie, čo sa môže stať, je reset virtuálneho stroja. (Predpokladáme bezpečnú

[2]Rozumejú sa zdroje ako napr. hardware, software

virtualizáciu PC.) Vyvíjať môže tiež bežný program pre iný OS, alebo pre inú hardwarovú platformu (napr. pre PowerPC) bez toho, aby musel fyzicky daný hardware vlastniť.

- **Poskytovateľ serverových riešení** nakúpi výkonný hardware, rozdelí jeden fyzický počítač na viacero virtuálnych a môže predávať "virtuálne servery" zákazníkom. Opäť v prípade bezpečnej virtualizácie si jednotlivé virtuálne stroje nezasahujú jeden druhému do zdrojov, sú vzájomne chránené, výpadok jedného z nich neovplyvní beh ostatných, atď...

- **Firma** môže svojim zamestnancom nainštalovať tzv. tenkých klientov (angl. thin client), ktorí sa pripájajú na centrálny server a celá práca zamestnanca prebieha vo virtuálnom prostredí na centrálnom serveri. Odpadá tým nutnosť starať sa o software na jednotlivých užívateľských PC a všetko sa spravuje na jednom mieste.

Nevýhody virtualizácie spočívajú v miernej strate výkonu, ktorý sa investuje do "réžie". Záleží od typu virtualizácie, či táto réžia bude len minimálna, alebo či bude podstatnou stratou výkonu.

5.1 Spôsoby virtualizácie

K virtualizácií môžeme pristupovať viacerými spôsobmi. Vždy záleží od našich očakávaní. Hrubé rozdelenie sme videli už v kapitole 5. Teraz sa pozrieme na problematiku z opačného pohľadu.

5.1.1 Platformová virtualizácia

Platformová virtualizácia sa snaží vytvoriť simulované prostredie pre beh hosťovského programu. Prostrediu sa hovorí "virtuálny stroj" a hosťovský program, ktorým je zvyčajne kompletný operačný systém, v ňom beží ako keby bol inštalovaný reálne na danej platforme. Takýchto virtuálnych strojov môže byť na jednom fyzickom počítači viacero. Ich počet je obmedzený len veľkosťou dostupných zdrojov (typické limitujúce faktory sú veľkost fyzickej RAM a pevného disku). Hosťovský OS bude vyžadovať prístup ku hardwaru, takže súčasťou virtualizácie sú aj virtuálne zdroje ako napr. pevný disk, sieťová karta, grafická karta. Všetky sa istým netriviálnym spôsobom mapujú na fyzické zariadenia hostiteľského počítača.

O koordináciu virtuálnych strojov sa stará špeciálny program, VMM alebo tiež hypervisor. Rieši najmä zdieľaný prístup ku fyzickým zariadeniam. Viac sa o VMM píše v časti 5.2.1

Požiadavky na platformovú virtualizáciu sú rôzne a teda aj prístupy ku jej realizácií sú viaceré:

Natívna virtualizácia vytvára kompletné prostredie virtuálneho stroja. Je vhodná pre beh operačného systému bez nutnosti jeho modifikácie. Virtualizované sú všetky potrebné súčasti počítača a hosťovský OS nemá (nemal by mať) šancu zistiť, že funguje vo virtuálnom prostredí. Skonštruovanie takéhoto virtuálneho stroja je z uvedených možností najťažšie a vyžaduje splnenie istých podmienok, preto je tomuto typu venovaná samostatná kapitola (5.2).

Väčšina inštrukcií je z dôvodu efektivity vykonávaná priamo na fyzickom procesore, čo nás obmedzuje na virtualizáciu rovnakej hardwarovej architektúry ako máme fyzicky v počítači.

Emulácia vytvára kompletné prostredie podobne ako pri natívnej virtualizácií. Rozdiel je v tom, že žiadna inštrukcia sa nevykonáva na hostiteľskom procesore priamo. Každý kus kódu sa softwarovo emuluje a to prináša možnosť prenositeľnosti emulácie aj na inú architektúru. Týmto spôsobom je riešená virtualizácia cudzej hardwarovej platformy, a je to zároveň jediný spôsob v tomto prípade.

Vykonávanie inštrukcií je rádovo pomalšie ako pri natívnej virtualizácií.

Paravirtualizácia prináša úplne odlišný prístup v porovnaní s predošlými riešeniami. Namiesto toho, aby sme sa snažili dokonale napodobniť všetky vlastnosti hardwarovej architektúry, urobíme menší ústupok. Budeme požadovať, aby bežiaci operačný systém nepristupoval priamo ku zdrojom virtuálneho stroja, ale aby komunikoval s VMM pomocou dohodnutého API v prípade potreby prístupu ku systémovým zdrojom. Takto si zabezpečíme, že všetky inštrukcie môžeme bez starostí vykonať priamo na hostiteľskom procesore bez ich analyzovania. Je však nutné, aby bol operačný systém modifikovaný pre beh pod paravirtualizáciou. Modifikácia spočíva v odstránení kritických častí, ktoré pristupujú ku HW, a namiesto toho je implementované volanie VMM cez API. K dispozícií sú takto upravené jadrá najbežnejších OS: Linux, MS Windows, *BSD.

Ku spomaľovaniu dochádza iba minimálne vďaka zbaveniu sa nutnosti analyzovať a emulovať inštrukcie. Problémom môže byť len pomalá komunikácia cez API medzi hosťovským OS a VMM.

Aplikačná virtualizácia sa sústredí na virtualizovanie prostredia pre aplikácie bežiace v operačnom systéme. Ide o vrstvu abstrakcie medzi OS

a aplikáciami, ktoré v tomto prípade nie sú spúšťané priamo, t.j. nepoužívajú zdroje OS. Jedná sa o "zmenšenú" verziu paravirtualizácie, avšak iba pre účely aplikácií, nie celých OS. Spúšťaná aplikácia pomocou OS API komunikuje s virtuálnym strojom, ktorý pre ňu zabezpečuje zdroje ako prideľovanie pamäti, prácu so súbormi, vstup a výstup... Typickým príkladom je Java na x86 platforme. Programy sú spúšťané pod Java Virtual Machine, ktorý robí prostredníka medzi OS a Java aplikáciou.

Výhody spočivajú v izolácií takto spúšťanej aplikácie a teda prináša istú mieru bezpečnosti pri vykonávaní. Virtuálny stroj totiž môže obmedzovať komunikáciu aplikácie s okolitým svetom (napr. na vybrané súbory alebo pridelí len isté množstvo pamäte...).

5.1.2 Virtualizácia zdrojov

Virtualizácia zdrojov je jednak súčasťou platformovej virtualizácie, kde sa zaoberá otázkou, ako riešiť prístup hosťovských OS vo virtuálnych strojoch ku fyzickým zdrojom počítača. Okrem toho to môže byť aj ľubovoľný iný postup, ktorým sa zakryjú fyzické charakteristiky zariadenia aj pre využitie mimo virtuálnych strojov.

Dôvod, prečo sa snažíme virtualizovať prístup ku zariadeniam je aj ten, že zariadenia neboli stavané na to, aby boli ovládané viac ako jedným operačným systémom naraz. S tým sa pri fyzickej konštrukcií nepočíta. Takisto OS predpokladá exkluzívny prístup ku zariadeniam.

Najbežnejší prípad virtualizácie zdrojov je rozdelenie pevného disku na menšie oddiely - partície. Je to spôsob dávno podporovaný prakticky v každom OS a umožňuje istý spôsob ochrany medzi partíciami.[3]

Pevné disky je možné združovať do väčších zväzkov a nechať ich navonok vystupovať ako jediný pevný disk s väčšou kapacitou prípadne väčšou odolnosťou voči fyzickému zlyhaniu. Typicky je tento prístup implementovaný vo volume manažéroch a RAID systémoch (či už SW alebo HW).

Pre účely virtualizácie je možné vyhradiť celý fyzický disk (alebo jeho partíciu) pre použitie vo virtuálnom stroji, kde sa tento bude tváriť ako fyzický disk. Taktiež je možné alokovať súbor dostatočnej veľkosti a použiť ho ako virtuálny pevný disk.

Sieťové pripojenie sa dá vyriešiť virtuálnou sieťovou kartou a navonok môže mať virtuálny stroj svoju vlastnú adresu na tej istej sieti, ako fyzický stroj. Taktiež je možné pre viaceré virtuálne stroje vyrobiť virtuálnu podsieť,

[3] Vírus vám zmaže celý disk C: ale ostatné partície ostanú nedotknuté

do ktorej budú zapojené a voliteľne úplne oddelené od fyzického sieťového spojenia.

Dve alebo viacero fyzických sieťových pripojení sa dajú kombinovať takže vystupujú ako jediné pripojenie s dvojnásobnou priepustnosťou prípadne spoľahlivosťou (odolnosť voči výpadku, pokiaľ vypadne niektoré z fyzických spojení).

Architektúra x86 od čias procesora 80386 umožňuje virtualizáciu operačnej pamäte pomocou technológie nazývanej stránkovanie (angl. paging). Aplikácie pri prístupe do pamäte používajú lineárnu adresu, ktorá v procesore podlieha prekladu na adresu fyzickú. Takto je možné oddeliť adresné priestory medzi aplikáciami. Je tiež možné prideliť aj pamäť, ktorá fyzicky nie je dostupná a pri prístupe zapisovať a čítať dáta z iného média (napr. pevný disk). Takýmto spôsobom je riešené swapovanie v OS.

Virtualizovať je možné aj výpočtovú silu pomocou prepojenia viacerých procesorov alebo celých počítačov do útvaru nazývaného "cluster". Cluster sa navonok javí ako jeden počítač, ale rozdeľuje zátaž medzi všetkých členov clustera, takže disponuje veľkým výpočtovým výkonom. Nie je pritom žiaden problém na takomto clusteri pustiť platformovú virtualizáciu a nechať na ňom bežať viacero virtuálnych počítačov. Takto sa vlastne reorganizuje resp. prerozdelí výpočtová sila jednotlivých účastníkov clustera (fyzické počítače) medzi virtuálne počítače.

5.2 Natívna virtualizácia

Natívnej virtualizácií venujeme samostatnú kapitolu práve pre relatívnu zložitosť nasadenia, ale aj z dôvodu, že kombinuje efektivitu vykonávania inštrukcií spoločne s kompletnosťou prostredia virtuálneho stroja. Systémy dokážu v natívnej virtualizácií bežať spravidla bez nutnosti upravovať ich.

Pri implementácií natívnej virtualizácie si môžeme úroveň efektivity zvoliť z celej škály počnúc emuláciou každej jednej inštrukcie, končiac emuláciou iba kritických inštrukcií.

5.2.1 Hypervisor - srdce virtualizácie

Hypervisor, alebo tiež VMM (Virtual Machine Monitor) - správca virtuálneho stroja - predstavuje jadro celej virtualizácie. VMM tvorí rozhranie medzi hostiteľským HW a hosťovským OS. Má na starosti virtualizáciu zdrojov pre virtuálny stroj, zapuzdrenie virtuálneho stroja aby bežal nad hostiteľským HW. Taktiež obsluhuje väčšinu výnimiek virtuálneho stroja a teda vytvára prostredie, ktoré pre hosťovský OS vyzerá ako keby bolo reálne.

Z hľadiska konkrétnej implementácie rozlišujeme 2 druhy VMM:

VMM 1. typu - Tento typ VMM beží priamo na hardwari hostiteľského počítača. Správa sa ako operačný systém. Má na starosti prideľovanie zdrojov, ovládače zariadení, rozdeľovanie výpočtového času medzi jednotlivé virtuálne stroje. Výhoda spočíva v úplnej kontrole nad fyzickými zdrojmi, čiže nad hostiteľským počítačom. Nemusí sa spoliehať na iný OS, lebo sám ním je. Toto prináša aj radu bezpečnostných výhod, lebo uplatňovanie bezpečnostnej politiky medzi virtuálnymi strojmi sa deje priamo vo VMM. Nevýhody spočívajú v nutnosti obsahovať ovládače zariadení, ktorých je vo svete obrovské množstvo. Táto nevýhoda sa dá kompenzovať "ušitím" VMM na mieru konkrétnemu systému, čím sa ovládače obmedzia len na zariadenia, ktoré v danom systéme sú. Takto sa na druhej strane stráca univerzálnosť a prenositeľnosť takéhoto VMM. Takéto VMM je možné nájsť aj ako súčasť jadra už existujúceho OS, kedy je možné využiť ovládače zariadení, ktoré sú pre tento OS už k dispozícií a sú dobre otestované. Je to vlastne kombinácia VMM 1. typu spolu s komfortom zrelého, prepracovaného OS. (Príkladom je KVM[4] nachádzajúce sa v jadrách OS Linux.)

Nasadzuje sa práve v oblasti, kde bezpečnostné požiadavky prevyšujú požiadavky na univerzálnosť riešenia.

VMM 2. typu - VMM 2. typu je vo forme aplikácie, ktorá beží v hostiteľskom operačnom systéme. Jeho úlohou je koordinácia virtuálnych strojov. Využíva služby OS, v ktorom sa nachádza, čím sa zjednodušujú nároky na konštrukciu takéhoto VMM najmä v oblasti ovládačov zariadení. V tomto prípade VMM používa bežné API v komunikácií s OS, ktorý mu prideľuje zdroje. VMM sa teda nemusí starať o implementáciu množstva ovládačov, túto robotu prenecháva OS, a preto ostáva prenositeľnosť na vysokej úrovni. Bezpečnosť riešenia je v tomto prípade nižšia, lebo sa spoliehame na bezpečnosť hostiteľského OS. Pokiaľ je v tomto OS nejaká bezpečnostná diera a potenciálny útočník by získal kontrolu nad OS, získal by nepriamo kontrolu aj nad virtuálnymi strojmi, keďže tým získava kontrolu nad zdrojmi.

Pre univerzálnosť riešenia je tento typ VMM preferovaný v bežnom nasadení všade tam, kde dokonalá bezpečnosť nie je podmienkou č.1. Toto je napr. prípad domáceho používateľa, prípadne systémového programátora (zmienení v kapitole 5). Pre tieto kategórie užívateľov je ponúkaná bezpečnosť postačujúca.

Virtualizácia fyzických zdrojov vyzerá tak, že VMM si vyberie nejaké existujúce zariadenie, a voči virtuálnym strojom prezentuje správanie takéhoto

[4]angl. Kernel Virtual Machine

zariadenia. Ako konkrétny príklad si môžeme popísať riadenie prístupu ku grafickej karte.

V počítači je fyzicky inštalovaná karta značky nVidia, a má nejaké vlastnosti. Tieto vie využiť driver hostiteľského OS, resp. VMM. VMM implementuje virtuálnu grafickú kartu značky Intel, ktorej vlastnosti dobre pozná a prezentuje ju virtuálnym strojom. Virtuálne stroje pri výstupe na grafickú kartu používajú driver ku karte Intel, a VMM spracúva tieto dáta, a následne ich transformuje pre použitie na fyzickej karte nVidia. Podľa toho, ktorý virtuálny stroj je aktívny ten bude mať možnosť (do istej miery) ovplyvniť, čo sa naozaj udeje na obrazovke. Výstupy zvyšných strojov sa akumulujú v pamäti VMM, a prejavia sa (zapíšu sa do fyzickej grafickej karty) až keď budú aktívne.

5.2.2 Popek & Goldbergove požiadavky

V roku 1974 boli sformulované požiadavky, postačujúce podmienky, ktoré by mala spĺňať architektúra k tomu, aby mohla byť efektívne virtualizovaná. Požiadavky sformulovali Gerald J. Popek a Robert P. Goldberg v [10].

Inštrukčná sada procesora sa delí na viacero typov inštrukcií, ktoré si teraz objasníme:

Privilegované inštrukcie - Sú to inštrukcie, ktoré vyvolajú výnimku, pokiaľ sú použité užívateľským procesom, a zároveň nespôsobujú výnimku, pokiaľ sú použité systémovým procesom.

Senzitívne inštrukcie - Vykonanie takejto inštrukcie ovplyvňuje alebo číta nejaký globálny stav procesora alebo systému. Sem spadajú inštrukcie pre I/O, prístup ku konfiguračným registrom a iným informáciám globálneho významu. Niektoré inštrukcie môžu vyvolať výnimku pri použití z užívateľského procesu, a niektoré nie.

Bežné inštrukcie - Inštrukcie, ktoré ovplyvňujú iba lokálny stav úlohy, registre pre všeobecné použitie, pamäť procesu, ... Patria sem najmä inštrukcie aritmetické, logické, inštrukcie pre presun dát, prácu s reťazcami.

Prvá požiadavka znie: Počítač je virtualizovateľný (dokážeme skonštruovať VMM), pokiaľ množina senzitívnych inštrukcií tvorí podmnožinu privilegovaných inštrukcií. Inými slovami, chceme, aby došlo ku výnimke pri každej senzitívnej inštrukcií volanej z užívateľského prostredia. Výnimka totiž môže byť smerovaná do VMM, ktorý ju obslúži. Obsluha spočíva v rozpoznaní inštrukcie, ktorá výnimku vyvolala a v odsimulovaní efektu vykonania danej inštrukcie.

Druhá požiadavka sa týka rekurzívnej virtualizovateľnosti. Touto sa ďalej nebudeme zaoberať.

Musíme si uvedomiť, že senzitívne inštrukcie by nemali byť vyvolávané bez povšimnutia VMM. Tak by sa totiž hosťovský OS mohol dozvedieť, že beží vo virtualizácií a to by mohlo ovplyvniť jeho ďalšie konanie. Príklad: Hosťovský OS pri svojom štarte vykonáva prepnutie z reálneho do chráneného módu. Toto sa deje zápisom do konfiguračného registra procesora. Zápis je privilegovaná senzitívna inštrukcia a jej efekt dokážeme zachytiť a odsimulovať. Ak by však čítanie z tohto registra nebola privilegovaná inštrukcia, OS by sa mohol dozvedieť skutočný stav fyzického systému bez toho, aby sa o tom VMM dozvedel. Znamená to, že by sa mohol dozvedieť (z jeho hľadiska) neplatné hodnoty.

5.2.3 Schopnosť virtualizovať platformu x86

Keď sa pozrieme na inštrukčnú sadu x86 a dôkladne ju preštudujeme, zistíme, že obsahuje vyše 17 inštrukcií, ktoré porušujú prvú požiadavku Popek & Goldberga. Medzi takéto inštrukcie patrí napr. SMSW, ktorá číta konfiguračný register CR0 (jeho spodných 16 bitov), nie je privilegovaná, ale je senzitívna. Patria sem aj inštrukcie SGDT, SLDT, SIDT, ktoré dokážu získať fyzické pamäťové adresy dôležitých systémových tabuliek. Tieto opäť sú globálne pre celý počítač a teda obsahujú niečo iné ako to, čo očakáva hosťovský OS. Obsahujú totiž hodnoty platné pre hostiteľský OS, a tie môžu byť odlišné.

Požiadavky Popek & Goldberga sú postačujúce, nie nutné, takže nám nezabránia v konštrukcií virtuálneho stroja. Problém so senzitívnymi inštrukciami sa dá vyriešiť. Jedna z najtriviálnejšie znejúcich možností (ale už menej triviálna čo do realizácie) je dynamické skenovanie pred vykonávaním. Spočíva v tom, že všetok hosťovský kód sa pred vykonaním podrobí analýze a ku nevhodným inštrukciám sa pripojí umelé vyvolanie výnimky, predanie riadenia VMM a následné odsimulovanie efektu vyvolania danej inštrukcie.

V praxi to znamená, že bežné inštrukcie sa nechajú vykonať bezo zmien, a problémové inštrukcie[5] skôr než sa dostanú narad, spôsobia vyvolanie výnimky. Efektivitu teda nestratíme nejako dramaticky, lebo stále väčšina inštrukcií bude vykonávaná bez zásahu VMM. Pre porovnanie, celkový počet inštrukcií x86 platformy sa pohybuje okolo 260-300, kdežto problémových je do 20, čo je v tomto prípade menej ako 10%. Navyše, výpočtovo náročné časti sú práve tvorené bežnými inštrukciami (aritmetika a prenášanie dát v pamäti).

Mierny pokles efektivity môže spôsobiť práve technika skenovania a ana-

[5]tie, ktoré sú senzitívne, a zároveň nie sú privilegované

lyzovania inštrukcií, keďže nie je možné vynechať žiadnu časť kódu. Popri tom si treba do pomocných dátových štruktúr ukladať informácie o častiach pamäti, ktoré už boli skenované. Takéto informácie sú zvyčajne tvorené poľom bajtov, kde každý bajt obsahuje informáciu o jednej skenovanej inštrukcií. Informácie zahŕňajú dĺžku inštrukcie a nutnosť simulácie. Pozor si treba dať aj na skutočnosť, že programy môžu modifikovať sami svoje inštrukcie v pamäti, čo by mohlo prekaziť analýzu, ak už bola daná časť skenovaná. Preto sa zavádza ochrana pamäťových stránok proti zápisu. Ak sa aplikácia pokúsi o zápis do časti, ktorá už bola analyzovaná, vyvolá sa výnimka, ktorú VMM dokáže obslúžiť. Obsluha dočasne umožní zápis, a následne vykoná opätovnú analýzu po ktorej opäť zápis znemožní.

5.3 Podpora virtualizácie v procesoroch

Ako sa ukázalo v sekcii 5.2.3, na platforme x86 existujú problémové inštrukcie, ktoré komplikujú natívnu virtualizáciu. Vyžadujú techniku dynamického skenovania pred vykonávaním, čo je veľmi náročné na implementáciu. Situácia by sa zlepšila, keby samotný procesor uľahčil virtualizáciu práve tým, že by detekoval problémové inštrukcie a spolupracoval s VMM pri ich vykonávaní.

V moderných procesoroch Intel existuje pre tento prípad technológia IVT, Intel Virtualization Technology, tiež "Vanderpool", a v procesoroch AMD mierne odlišná, ale principiálne rovnaká technológia AMD-V, tiež "Pacifica".

IVT prináša rozšírenie inštrukčnej sady o inštrukcie pre podporu virtualizácie, pod súhrnným názvom VMX, Virtual Machine Extensions.

5.3.1 VMX - Virtual Machine Extensions

VMX - Virtual Machine Extensions alebo tiež rozšírenie pre podporu virtuálnych strojov predstavuje účinný prostriedok pri podpore virtualizácie. Je to podpora priamo v procesore a dá sa kategorizovať do skupiny "natívna virtualizácia platformy Intel 32, resp. Intel 64". Technológia VMX podporuje beh VMM (Hypervisora), ktorý disponuje fyzickými prostriedkami počítača a koordinuje beh jedného alebo viacerých hosťovských počítačov.

Základnou myšienkou VMX je, aby čo najviac kódu hosťovského systému bežalo bez modifikácie. Pokiaľ by mal hosťovský systém vykonať inštrukciu, ktorá číta alebo upravuje globálny stav systému, dôjde k jeho pozastaveniu a riadenie sa predá do VMM. Tu sa vzniknutý stav rieši, a následne VMM vráti kontrolu hosťovskému systému.

VMX rozlišuje 3 stavy fungovania procesora:

- VMX je neaktívne, všetok kód beží na fyzickom procesore.

- VMX root mód, inštrukcie vykonáva VMM, má prístup ku všetkým zdrojom počítača, koordinuje činnosť virtuálnych strojov.
- VMX non-root mód, riadenie má hosťovský systém vo virtuálnom stroji a každá podozrivá inštrukcia je hlásená návratom do root módu.

Činnosť VMX sa začína volaním inštrukcie VMXON a procesor tak prechádza do VMX root módu. VMM následne inicializuje virtuálne stroje, ktoré bude obsluhovať, každému virtuálnemu stroju prislúcha v pamäti štruktúra VMCS (Virtual machine control structure). VMM vykoná inštrukciu VMLAUNCH, čím predá riadenie virtuálnemu stroju. VMX sa tak prepne do non-root módu. Hosťovský systém vo virtuálnom stroji vykonáva inštrukcie až dokým nespôsobí VM-exit, opustenie non-root módu, návrat do root módu a predanie riadenia VMM. VMM sa dozvie príčinu exit-u, vykoná príslušné opatrenia (emuluje efekt nevhodných inštrukcií), a vráti riadenie virtuálnemu stroju pomocou inštrukcie VMRESUME. Toto sa deje dokola, až sa eventuálne VMM rozhodne ukončiť virtuálny stroj, a zruší VMCS štruktúru daného stroja. Pre opustenie činnosti VMX môže použiť inštrukciu VMXOFF a tým prejde do stavu "VMX je neaktívne".

Príčiny pre pozastavenie hosťovského systému (VM-exit) a predanie riadenia VMM sú najmä: Pokus o čítanie alebo zápis kontrolných registrov CRx, ladiacich registrov DRx, špecifických registrov MSR. Ku VM-exit dochádza tiež pri pokuse o komunikáciu cez I/O porty použitím inštrukcií IN/OUT. VM-exit je spôsobený aj príchodom externého prerušenia, vznikom výnimky alebo softwarovo vyvolaným prerušením počas VMX non-root módu. V každom prípade VMM má k dispozícií kód, zhruba popisujúci príčinu exit-u, a tiež pole bitov s podrobnejšími informáciami ku danej príčine.

Pomocou VMX dokážeme vytvoriť virtuálny stroj a nechať v ňom bežať operačný systém bez nutnosti nejako ho upravovať. Hosťovský systém má k dispozícií všetky 4 úrovne CPL, keďže VMM funguje v úplne oddelenom prostredí a nemusí obsadzovať úroveň CPL=0. Systém vo vnútri virtuálneho stroja nemá žiadny spôsob, ako by sa mohol dozvedieť, že beží vo virtualizácií. Snaha o prečítanie konfiguračného registra, bude isto viesť ku VM-exit a VMM môže vrátiť ľubovoľne upravené výsledky. Nepomôže ani metóda čítania registra TSC, ktorý obsahuje časové informácie (počet tikov od resetu). Čítanie TSC môže vrátit vopred upravené hodnoty podľa ľubovôle VMM. Prepínanie medzi root a non-root módom nie je pre hosťovský systém viditeľné. Dochádza pri tom ku takmer úplnej výmene stavu procesora. Keďže miesto pre ukladanie stavov je vopred definované (nachádza sa v štruktúre VMCS), procesor môže robiť prepnutie veľmi efektívne.

VMX má obmedzené použitie, dokážeme priamo virtualizovať iba režimy IA-32e, 32-bit chránený režim a Virtual-8086. Reálny mód nie je podporo-

vaný, a je treba jeho inštrukcie buď emulovať softwarovo vo vnútri VMM, alebo využiť Virtual-8086 režim za asistencie V86 monitoru. Všetky okrajové situácie, akými je napríklad prepínanie režimov, je nutné vyriešiť vo VMM.

5.3.2 Virtual-8086 mód po druhý krát

Súčasťou všetkých procesorov Intel 80386 a vyšších je režim Virtual-8086 (V86), ktorý umožňuje virtualizovať prostredie procesora Intel 8086. Tento režim je vhodný pre spúšťanie programov písaných pre reálny mód pod chráneným režimom.

Princíp fungovania je podobný ako pri virtualizácií pomocou technológie VMX. 8086 kód beží ako samostatná úloha (task) chráneného režimu. Súčasťou virtualizácie je V86 monitor (VMM), ktorý koordinuje beh V86 úlohy. V86 úloha má k dispozícií reálny spôsob adresácie, 1 MB fyzickej pamäte, 16-bit prostredie. Ak sa V86 úloha pokúsi vykonať inštrukciu zasahujúcu do globálneho stavu procesora, signalizuje sa výnimka do chráneného režimu, a bude obslúžená vo VMM.

Postup použitia je nasledovný: Hostiteľský systém zavedie do pamäte kód, ktorý chce vykonať, vytvorí 32-bit TSS, ktorý bude reprezentovať budúcu V86 úlohu, udržovať jej stav procesora. Vstup do V86 sa vykoná volaním TSS prípadne IRET návratom do TSS, pričom sa nastaví príznak EFLAGS-.VM=1. Nastavenie tohto príznaku musí byť súčasťou prepnutia úloh. Nedá sa zmeniť priamo. Úloha V86 následne beží až do chvíle, kedy sa pokúsi vykonať nejakú chránenú inštrukciu, softwarové prerušenie, alebo do príchodu externého prerušenia, prípadne výnimky procesora. Vtedy procesor prepne úlohy naspäť do chráneného režimu a predá riadenie do VMM. VMM sa dozvie príčinu návratu, vykoná príslušné opatrenia (emuluje efekt nevhodných inštrukcií), a vráti riadenie V86 úlohe, alebo sa rozhodne ukončiť ju.

V systéme môžu naraz fungovať viaceré úlohy V86. Každá V86 úloha má k dispozícií priestor lineárnych adries 0 - 0x10FFEF, t.j. 1 MB + 64 kB. Pomocou stránkovania dokážeme lineárne adresy každej úlohy mapovať do rôznych častí fyzickej pamäte. Pri prepínaní úloh sa dá vymeniť obsah registra CR3, ktorý obsahuje ukazovateľ do adresára stránok, a tým obmeniť mapovanie stránok.

Praktické využitie V86 spočíva napríklad v možnosti spúšťať starý software pôvodne určený pre MS-DOS v okne moderného operačného systému ako Windows alebo GNU/Linux.

6 Program Prot386

Princípy fungovania chráneného režimu, hoci sú presne definované, predstavujú netriviálny problém pri snahe pochopiť ich z obyčajného textového výkladu. Študent si môže vybrať technickú úroveň spracovania - na webe je veľa návodov od tých najneformálnejších až po oficiálne podklady od firiem Intel alebo AMD. Ak chce človek úplne pochopiť danú problematiku, strávi študovaním nemalé množstvo času.

Pre lepšie pochopenie niektorých dejov môže poslúžiť program, v ktorom by si záujemca mohol sám nastaviť niektoré hodnoty a sledovať, ako sa systém ochrany zaspráva. Túto úlohu sa snaží plniť program Prot386.

Funkcie Prot386:

- Vytváranie a modifikovanie deskriptorov a ich prehľadné zobrazenie v tabuľkách GDT, LDT a taktiež IDT.
- Podpora viacerých LDT. Je možné prepínať si medzi aktívnymi LDT.
- Spustenie kódu, ktorý demonštruje nejakú funkciu. Napríklad naplnenie CS segmentu pomocou volania alebo prístup ku dátovému segmentu. Všetky parametre si užívateľ nastaví a využijú sa deskriptory, ktoré si vopred vytvoril.
- Obsluha všetkých výnimiek, ktoré môžu nastať - okamžite sa zobrazí varovné okno o nevydarenej akcii.
- Stránkovanie a možnosť upravovať tabuľky stránok a tým meniť mapovanie lineárnej pamäte.
- Hex-editor pamäte, pričom počiatočnú adresu je možné zadať troma spôsobmi.
- Nástroj na prepočítavanie pamäťových adries medzi rôznymi úrovňami mapovania.
- Výpis podrobných informácií o stave dôležitých registrov procesora a stave zásobníka v prípade výnimky.
- Ochrana pamäte jadra a segmentov pred neodborným zásahom užívateľa. Ochrana sa dá aj vypnúť pre prípad, keď užívateľ nechce byť obmedzovaný v experimentovaní (súčasťou vypnutia ochrany je aj upozornenie na možné dôsledky).

- Technologické lahôdky priamo v zdrojovom kóde ako napríklad obsluha výnimky v samostatnej úlohe (Task-gate), zistenie mena procesora, detekcia dostupnej pamäte RAM, testovanie funkčnosti brány A20.

Program nie je simulátor. Dané javy vykonáva priamo na procesore počítača, kde beží. Z tohto dôvodu nie je možné, aby bežal ako aplikácia v operačnom systéme MS Windows, GNU/Linux a iných, keďže tieto bežia v chránenom móde a Prot386 by nemohol priamo pracovať s pamäťou a deskriptormi. Tento program funguje sám ako operačný systém a spúšťa sa pri štarte počítača z USB-kľúča, CD-ROM média alebo iného zariadenia schopného boot. Požiadavky na OS teda niesú žiadne, program si môže vyskúšať každý, kto má vo svojom počítači procesor Intel 80386 alebo novší, kompatibilný s rodinou Intel (AMD, Cyrix, ...). Pre spustenie programu stačia 2 MB RAM, pre plnú funkčnosť je odporúčané mať aspoň 8 MB.

Veľký dôraz bol kladený na bezpečnosť počítača. Program nič nezapisuje na pevný disk, ani na iné I/O zariadenie s výnimkou grafickej karty. Pokiaľ dôjde ku výnimke, ktorá by mohla poškodiť stav PC (nemalo by sa stávať bežne), program okamžite ukončí činnosť a pre istotu reštartuje počítač.

Procesor sa pri štarte programu nachádza v reálnom móde. Program si vyrobí potrebné štruktúry v pamäti a následne sa prepne do chráneného módu, kde pracuje po celý čas, až do ukončenia a resetu - vtedy sa prepína opäť do reálneho módu. Obidve akcie boli popísané v [1].

Okrem "hrania sa" v hotovom prostredí, program slúži aj ako návod, ako takéto aplikácie písať - zdrojové kódy sú k dispozícií pod licenciou GNU GPL.

6.1 Bootloader

Aby mohol byť program Prot386 spustený, treba ho nejakým spôsobom zaviesť do pamäte. K tomuto účelu slúži bootloader, ktorý je súčasťou programu. Aby sme boli presní, bootloader je samostatný program o veľkosti 512 B, ktorý je pripojený na začiatku samotného jadra Prot386. Celkovo je teda zavádzaný program "zlepený" z dvoch programov. Pre zavedenie programu do pamäte je nutné vedieť pár detailov, ako to v PC funguje.

Po zapnutí PC a vykonaní inicializačných rutín BIOS-u sa procesor dostáva do reálneho módu. BIOS sa snaží nabootovať z vybraného zariadenia a to takým spôsobom, že z neho prečíta prvých 512 B (0. sektor - boot blok), tieto uloží do pamäte na lineárnu adresu 0x7C00. Ak sa posledný word celého sektora rovná magickej konštante 0xAA55, považuje sa boot blok za platný a riadenie je odovzdané na lin. adresu 0x7C00. V opačnom prípade BIOS skúša ďalšie zariadenie v poradí, až kým sa mu nepodarí nabootovať (alebo

kým nevypnú prúd). Ak bol úspešný, na adrese 0x7C00 už čaká boot blok, ktorý sa má postarať o naštartovanie operačného systému. V našom prípade bol na začiatku média bootloader. Jeho úlohy sú nasledovné:

- Vypísať na obrazovku niečo v štýle "štartuje sa systém" - toto sa vykoná pomocou volania služby BIOS-u, ktorá zapisuje text na obrazovku (int 0x10).
- Načítať zvyšné sektory z média na vhodné miesto do pamäte - opäť služby BIOS-u (int 0x13) a trocha aritmetiky so segmentami v pamäti a sektormi na diskete. Za povšimnutie stojí fakt, že BIOS pri boot z CD alebo USB-kľúča emuluje klasickú 1.44 MB disketu, čo nám zjednodušuje celý kód.
- Prepnúť sa do chráneného režimu a predať riadenie vstupnému bodu programu Prot386.

6.1.1 Prepnutie do chráneného módu

Nasledujú úkony súvisiace s prepnutím do chráneného módu.

- Povoliť hradlo A20.
- Premapovať hardwarové prerušenia zápisom na porty radiča prerušení - obvod 8259.
- Vytvoriť tabuľku GDT a v nej aspoň 3 deskriptory: 0. Neplatný deskriptor, 1. Kódový segment, 2. Dátový segment a zásobník.
- Naplniť register GDTR lineárnou adresou novovytvorenej tabuľky.
- Nastaviť bit PE (Protection Enable) v registri CR0.
- Vykonať skok do nového segmentu, presnejšie na selektor kódového segmentu z tabuľky GDT. Tým sa vyčistí fronta predpripravených inštrukcií z vyrovnávacej pamäte a procesor sa prepne do 32-bitového chráneného módu.

Skok do nového segmentu nie je len tak na náhodnú adresu, ale práve na adresu, kam bootloader zaviedol zvyšný kód - jadro systému. Tým sa práca bootloadera skončila a spolu s ním aj reálny mód a 16-bitová adresácia. Toto všetko sa musí zmestiť do 512 B.

6.1.2 Návrat do reálneho módu

Táto akcia sa vykoná pri ukončení programu, hoci nie je nutná. Reštart PC sa dá vykonať aj z chráneného módu a po reštarte procesor automaticky prejde do reálneho módu. Keďže prepnutie do reálneho módu je z programátorského hľadiska zaujímavá úloha, bola zaradená do zdrojového kódu minimálne z edukačných dôvodov.

6.1.3 Hradlo A20

Pokiaľ program pracuje v chránenom móde, je nutné mať A20 povolené, inak každá operácia s pamäťou nad 1 MB vyvolá pravdepodobne chybu. Detaily o hradle A20 sú spomenuté v [1].

Ovládanie A20 na počítačoch PC nie je štandardizované a dá sa vykonať množstvom rôznych spôsobov. Žiaden z nich nemusí byť spoľahlivý, hoci na novších počítačoch už problémy nenastávajú. Program Prot386 využíva najrozšírenejší postup - cez porty radiča klávesnice. Keďže fungovanie A20 je pre náš program kritické, následne sa overí jej funkčnosť zápisom a prečítaním viacerých hodnôt do / z pamäte. Ak test zlyhá, vypíše sa o tom hlásenie a program ďalej nepokračuje, aby sa predišlo nepredvídateľnému správaniu. Toto by za normálnych okolností nemalo nastať a indikuje to buď veľmi neštandartný počítač, alebo HW problém.

6.2 Použitie programu

Program bol písaný s dôrazom na jednoduché ovládanie. V menu sa dá hýbať šípkami hore a dolu. Výber sa potvrdzuje klávesou ENTER a návrat o úroveň vyššie vždy zabezpečí klávesa ESC. Okrem toho sa v niektorých oknách dajú využiť šípky vľavo a vpravo pre presúvanie sa medzi záznamami v tabuľke.

Rozhranie celého programu je v anglickom jazyku. Tento jazyk bol zvolený najmä kvoli technickým výrazom, ktoré tvoria značnú časť textov v programe. Slovenčina by v tomto prípade uberala na zrozumiteľnosti. Verím, že používateľ nebude mať problémy so základným porozumením.

6.2.1 Segmentácia

V menu pre segmentáciu máme možnosť prehliadnuť si obsah systémových tabuliek GDT, LDT, IDT. Deskriptory v každej tabuľke sa dajú upravovať. Niektoré sú chránené pred upravovaním kvôli stabilite programu. Bližší popis a zoznam chránených častí sa dá nájsť v sekcií 6.2.7. Pri každom deskriptore sú vypísané jeho relevantné atribúty, a taktiež aj jeho binárna podoba. Pri prechode jednotlivými atribútmi sa v binárnom poli zvýrazňujú časti,

ktoré kódujú daný atribút. Je možné upravovať aj samotný binárny reťazec a zmeny sa okamžite prejavia vo vypísaných atribútoch. Pohyb medzi deskriptormi zabezpečujú šípky vľavo a vpravo.

Tabuľka GDT je jediná v celom systéme a obsahuje deskriptory globálne platné pre všetky procesy. Vidno v nej teda aj deskriptory, ktoré využíva samotný program Prot386 ku svojej činnosti. Tieto sú chránené pred modifikáciou zo strany užívateľa.

V tabuľke LDT sa nachádzajú deskriptory platné len v obmedzenom rozsahu. Tabuliek LDT môže byť viac, a voľba tej aktuálnej sa deje zápisom do registra LDTR. Zvyčajne sa pri prepínaní úloh bežiacich v systéme (procesov) vymení aj tabuľka LDT, aby každý proces mal svoju vlastnú, s ktorou môže pracovať. Užívateľ si môže v GDT vytvoriť ďalšie deskriptory pre LDT, a následne plniť register LDTR ich adresami, t.j. vyberať si aktívnu LDT.

Tabuľka IDT obsahuje volacie brány pre obslužné rutiny prerušení. Slúži skôr ako doplnok a nie je dôvod modifikovať jej obsah. Nie je to však zakázané, takže sa dá experimentovať s prerušeniami.

6.2.2 Stránkovanie

Cez položku menu "Page Directory" sa dostaneme do adresára stránok (tabuľka 2. úrovne). Záznamy (PDE, page directory entry) v tejto tabuľke môžeme upravovať podobne ako deskriptory v segmentácií. Pre niektoré hodnoty sa vzťahuje ochrana. Adresár stránok obsahuje najmä fyzické adresy tabuliek stránok. Z každého záznamu sa dá otvoriť tabuľka stránok (page table, tabuľka 1. úrovne). Tu sa nachádzajú záznamy PTE, page table entry. Majú podobnú štruktúru ako PDE. Každý záznam popisuje stránku v pamäti, najmä jej fyzickú adresu a prístupové práva.

6.2.3 Vykonávanie kódu

Akčná časť programu sa nachádza práve tu. Užívateľ si vyberie modul, ktorý chce spustiť, nastaví mu oprávnenia a následne ho vykoná. K dispozícií sú tri moduly: Vyvolanie softwarového prerušenia, prístup ku lineárnej adrese, a ukázanie dosiahnutého CPL. Modul sa spúšťa ako samostatná úloha. Keď nastane akákoľvek výnimka, dozvieme sa o nej, a následne bude úloha zrušená a riadenie bezpečne vrátené jadru.

V skutočnosti vykonanie vyzerá nasledovne:

- Do pracovného segmentu sa nahrá kód modulu. Pre jednoduchosť sa tam skopíruje celé jadro. Pracovný segment si vyberáme v nastaveniach. Mal by to byť kódový segment alebo brána ukazujúca na kódový segment.

- Vytvorí sa štruktúra pre beh úlohy (TSS, task state segment). Naplní sa ukazovateľom na prvú inštrukciu (ukazuje do pracovného segmentu), zásobníkmi pre všetky úrovne CPL, počiatočnou úrovňou CPL (z nastavení).

- Vykoná sa CALL resp. JMP inštrukcia na vytvorené TSS. To spôsobí prepnutie úlohy v procesore, a začne sa vykonávať kód modulu.

- Modul ukončí svoju činnosť inštrukciou IRET, ktorá vráti riadenie volajúcej úlohe, t.j. jadru. Ak počas behu došlo ku výnimke, obsluha výnimky ukončí vykonávanie úlohy opäť pomocou IRET.

V TSS štruktúre sa používajú niektoré pomocné deskriptory segmentov, tieto majú indexy 250-254 v GDT tabuľke.

6.2.4 Hex-editor

Ďalšia akčná súčasť programu. Cez hex-editor sa dostaneme priamo do pamäte RAM, a máme možnosť upravovať bajty v hexadecimálnej sústave. Výber adresy sa dá uskutočniť troma rôznymi spôsobmi. Dá sa zadať počiatočná logická, lineárna alebo fyzická adresa. Keďže program k pamäti vždy pristupuje len pomocou logickej adresy, ostatné typy adries sa prepočítajú na logickú adresu (pokiaľ existuje také mapovanie).

V pamäti je možné upraviť si niektoré bajty, zmeniť nastavenie segmentov prípadne stránkovania, vrátiť sa, a skúmať zmeny v mapovaní fyzickej pamäte na to, čo vidíme (logický adresný priestor).

6.2.5 Obsluha výnimiek

V sekcií "Interrupt Handlers" vidíme zoznam výnimiek, ktoré procesor generuje. Môžeme si ktorúkoľvek výnimku umelo vyvolať. Tým sa zavolá obsluha danej výnimky. Toto slúži len na otestovanie, ako sa daná obsluha zaspráva.

Všimnime si, že štandartne obsluha vypíše okno so stručným popisom výnimky, okno so stavom dôležitých registrov procesora a okno s výpisom zo zásobníka. Na zásobníku sa v prípade ozajstnej výnimky nachádza kód chyby, offset a segment inštrukcie pri ktorej došlo ku výnimke, stav EFLAGS registra. Tieto informácie sa dajú využiť za predpokladu, že poznáme adresy funkcií v pamäti. Potom môžeme určiť, v akej funkcií, kde presne sa nachádza problémová inštrukcia. Adresy funkcií ochotne prezradí kompilátor pri zostavovaní programu a zapíše ich do súboru system.map.

6.2.6 Ostatné funkcie

Prot386 obsahuje zopár ďalších drobností, ktoré pomáhajú pri používaní programu samotného. Patria sem:

- Konvertor medzi desiatkovou a hexadecimálnou sústavou pre rýchle prepočty medzi sústavami v oboch smeroch. Princíp fungovania veľmi podobný eurokalkulačke.
- Prepočet adries medzi priestormi logických, lineárnych a fyzických adries. Tu sa rýchlo dá zistiť, ako sa prejavilo nastavenie segmentov a stránok.
- Typ procesora a veľkosť pamäte RAM - čisto z informatívnych dôvodov.
- Volanie inštrukcie CPUID s ľubovoľným parametrom. Slúži na získanie množstva rôznych informácií o procesore.

6.2.7 Ochrana systému

Systém ochrany v procesore chráni jadro pred procesmi na nižšej úrovni CPL. Kto však ochráni jadro pred užívateľom? Keďže Prot386 dáva užívateľovi prístup ku dôležitým štruktúram procesora, mohlo by sa neopatrným zásahom stať, že niektorú štruktúru zruší a spôsobí v lepšom prípade pád programu a reset procesora. V horšom prípade vykonanie náhodných inštrukcií a v katastrofickom scenáriu aj výstup na I/O zariadenie, napr. HDD. Aby sa riziko minimalizovalo, jadro programu sa chráni tak, že niektoré úkony užívateľovi nepovolí, a dá o tom aj náležite vedieť. Ochrana sa vzťahuje na nasledovné veci:

- Fyzická pamäť, v ktorej sa nachádza jadro programu a jeho štruktúry nesmie byť priamo modifikovaná. Aktuálne prvé 2 MB.
- Deskriptory 0-6 z tabuľky GDT, ktoré sú využívané jadrom, nesmú byť modifikované.
- Mapovanie (stránkovanie) lineárnych adries na fyzické adresy v rozsahu 0-2 MB musí ostať identické, t.j. 1:1.
- Nie je dovolené vytvoriť deskriptor, ktorý by sprístupňoval lineárne adresy z chráneného rozsahu 0-2 MB.

Takáto ochrana je veľmi účinná, pokiaľ predpokladáme, že samotný program Prot386 je napísaný bezchybne a sám od seba nespôsobí zlobu v počítači, čo sa zrejme zaručiť ani nedá, ale dá sa tomu minimálne veriť.

Pre ľudí, ktorí chcú experimentovať bez obmedzení a sú ochotní zobrať na seba riziko nepredvídateľného správania sa programu, je možnosť celú softwarovú ochranu jadra vypnúť. Pred vypnutím sa zobrazí varovné okno, ktoré má za úlohu odradiť užívateľa od takého kroku. Kto však chce experimentovať, tomu sa už ďalšie obmedzenia nekladú.

6.3 Technické detaily

Jadro prograu Prot386 je zostavené z viacerých zdrojových súborov v jazykoch assembler a C. Spolupráca oboch jazykov je bezproblémová. Z asm je možné volať funkcie C, taktiež aj opačne. Stačí, ak je dodržaná C volacia konvencia. Parametre funkcie uloží volajúci na zásobník v poradí z prava do ľava. Počas behu funkcia môže modifikovať registre EAX, ECX, EDX. Pokiaľ potrebuje aj ďalšie, musí ich hodnotu pred skončením obnoviť. Návratová hodnota sa uloží do EAX a funkcia vráti riadenie pomocou ret. Volajúci musí odstrániť parametre zo zásobníka (zvyčajne iba zvýši hodnotu registra ESP o veľkosť parametrov).

Pri písaní funkcií v asm si treba dať pozor na niekoľko skutočností. Treba zabezpečiť, aby dátový segment a segment zásobníka (DS, SS) ukazovali na deskriptory s rovnakou bázou. Najjednoduchšie je použiť ten istý deskriptor pre oba segmenty. Predávanie parametrov funkciám sa totiž na to spolieha. To isté platí aj o báze kódového a dátového segmentu (CS, DS), keďže celý program beží v plochom (flat) pamäťovom modeli. Ďalší problém spočíva v tom, že gcc kompilátor môže generovať volania funkcií memcpy, memmove, ..., čo je aj spomenuté v manuálových stránkach. Za normálnych okolností to nie je problém, lebo ku výslednému programu sa linkuje štandartná knižnica libc, ktorá tieto funkcie implementuje. V našom prípade sa libc nepoužíva, lebo by zbytočne zväčšila veľkosť programu. V praxi sa vyskytlo len volanie funkcie memcpy (pri priraďovaní a kopírovaní C štruktúr), a z toho dôvodu je memcpy v programe implementovaná.

Všetky moduly sú preložené buď pomocou gnu *gcc* alebo gnu *as* do podoby objektových súborov. Tie sú nakoniec zlinkované na adresu 0x10000 (toto je adresa, kam bootloader celé jadro zavedie). Výsledný súbor je v binárnom formáte, neobsahuje žiadne hlavičky spustiteľného súboru. Vstupným bodom je prvá inštrukcia jadra - funkcia _start umiestnená v module main.s. Jadro by nemalo prekročiť veľkosť 576 kB, inak nastanú problémy s jeho zavádzaním do pamäte v reálnom móde.

Bootloader je prekladaný a linkovaný zvlášť kompilátorom *as86* a linkerom *ld86*, ktoré vytvárajú 16-bitový kód. Výsledný binárny kód - 512 B dlhý bootloader je pripojený na začiatok súboru s jadrom. Takto vzniknutý súbor (Image) môže byť zapísaný na disketu alebo iné médium ako obraz.

Vstupy z klávesnice sú čítané cez I/O port klávesnice, keď je treba prečítať vstup od užívateľa, program začne v slučke kontrolovať, či bol stlačený kláves a postupne ich preberie a spracuje.

Výstup na monitor sa deje v textovom režime 80x25 znakov, ktorý je štandartne nastavený po zapnutí PC. Textová obrazovka je prístupná cez pamäť od lineárnej adresy 0xB8000. Znaky sa striedajú s informáciami o farbe každého znaku. Poloha kurzora sa nastavuje cez zápis na I/O porty grafickej karty.

7 Záver

Pre lepšiu kontrolu nad prístupom ku fyzickej pamäti existuje v procesoroch 80386 a vyšších podpora stránkovania pamäte. Pomocou stránkovania sa dajú riešiť prístupové práva ku častiam pamäte, premapovanie nesúvislých častí do súvislých blokov lineárnych adries, úplné oddelenie fyzických častí pamäte medzi procesmi pomocou jednoduchej výmeny tabuliek a v neposlednom rade ponúka aj virtuálnu pamäť pomocou naťahovania stránok z externého zdroja (napr. HDD) až v momente, keď je to potrebné.

Virtualizácia platformy Intel 32 resp. Intel 64 je na moderných procesoroch podporovaná technológiou VMX. Táto technológia umožnuje nechať vykonávať kód virtuálneho stroja bezo zmien, pritom zabezpečuje, že privilegované a aj senzitívne inštrukcie budú zachytené, a VMM sa nemusí starať o ich dynamické vyhľadávanie a modifikáciu kódu. Na rozdiel od bežnej natívnej virtualizácie netreba pri použití VMX mapovať celý hosťovský systém do úrovne CPL=3, pretože VMM v tomto prípade je úplne oddelené, a pohodlne dostáva riadenie až vtedy, keď sa vyskytne podozrivá inštrukcia.

Význam jednotlivých zmien v systéme ochrany v režimoch IA-32e vidím najmä v zjednodušení správy pamäte bez toho, aby bola obetovaná významná časť funkcionality. Zánik segmentácie znamená koniec rozumného využitia pre úrovne oprávnenia CPL 1 a 2. V stránkovaní sa totiž používajú len úrovne 0 a 3. Toto však neuberá na funkcionalite. Ďalej bolo zrušené hardwarové prepínanie úloh, čo tiež nemá významnejší dopad na funkcionalitu. Prepínanie úloh sa kvôli efektivite v operačných systémoch aj tak väčšinou rieši softwarovo.

Náhradou za chýbajúcu možnosť obsluhovať prerušenia prepnutím ulohy (cez task gate) je IST (interrupt stack table). Takto sa dajú obslúžiť aj prerušenia, ktoré vznikli z dôvodu chyby na zásobníku pre CPL 0.

Program Prot386 mal za cieľ uľahčiť záujemcom pochopenie problematiky chráneného režimu a celkovo fungovania procesorov z rodiny Intel. Na jednej strane umožňuje vyskúšať si niektoré deje v praxi, ponastavovať si vlastné hodnoty a pozrieť sa "čo to spraví". Na druhej strane poskytuje cenné informácie v podobe zdrojových kódov v jazykoch assembler, C, ktoré umožnia začínajúcemu systémovému programátorovi pochopiť nielen "ako to funguje", ale aj "ako to naprogramovať", resp. "ako s tým začať". Verím, že spomínaný cieľ sa podarilo naplniť.

Slovník pojmov

Adresácia - Spôsob prepočtu logických adries (to, čo používa programátor / užívateľ) na fyzické adresy (tak ako sú naozaj v pamäti). Nie nutne 1:1.

Adresár stránok - Štruktúra obsahujúca ukazovatele na tabuľky stránok.

Bázová adresa - Adresa, zvyčajne získaná zo segmentu, od ktorej sa pohybujeme v pamäti pripočítavaním offsetu.

Deskriptor segmentu - 8 bajtov dlhý záznam obsahujúci informácie o segmente ako napr. bázová adresa a limit, ale aj typ segmentu a iné.

Fyzické zariadenie - Kus hardware, ktorý existuje, t.j. leží v hostiteľskom počítači a dá sa uchopiť.

GDT/LDT - Globálna / Lokálna tabuľka deskriptorov[6] je tabuľka obsahujúca jednotlivé deskriptory. Lokálna tabuľka môže byť pre každú úlohu (task) iná. Pamäť je dostupná len cez segmenty, ktorých deskriptory sú uvedené v týchto 2 tabuľkách.

Hostiteľský OS - Operačný systém, ktorý beží na fyzickom zariadení a má kontrolu nad jeho zdrojmi. Zvyčajne ich poskytuje pre VMM a bežia v ňom viaceré hosťovské OS. Anglicky: Host OS.

Hosťovský OS - Operačný systém bežiaci vo virtuálnom stroji, vovnútri hostiteľského OS. Anglicky: Guest OS.

Hradlo A20 - Zariadenie pre kompatibilitu s 8086 procesorom. Ak je povolené, nerobí nič. Ak je zakázané, nuluje 20. adresnú linku procesora. Fyzická adresa = fyzická adresa AND 0xFFEFFFFF.

Hypervisor - Správca virtuálneho stroja. Viď VMM.

IDT - Tabuľka deskriptorov prerušení. Môže obsahovať až 256 deskriptorov, ktoré popisujú kódový segment s obslužnou rutinou pre dané prerušenie. Deskriptory tu uložené majú podobnú štruktúru ako deskriptor volacej brány.

Kanonická adresa - Lineárna adresa je v kanonickom tvare, pokiaľ používa iba toľko bitov, koľko implementuje daná architektúra. Aktuálne je to 48 bitov, takže bity 0-47 sú k dispozícií a bity 48-63 musia byť nastavené

[6] angl. Global / Local Descriptor Table

všetky na 0 resp. všetky na 1. Kontrola kanonickosti sa uplatňuje v režimoch IA-32e na mnohých miestach a prakticky vždy pred použitím lineárnej adresy.

Limit - Najväčší povolený offset v danom segmente. Vzťahuje sa ku konkrétnemu segmentu.

Mód - viď Režim.

Multitasking - Beh viacerých aplikácií naraz. Dosahuje sa rýchlym striedaním aktuálne vykonávaného kódu.

Offset - Časť logickej adresy, pripočítava sa ku bázovej adrese aby sme získali lineárnu adresu.

PAE - Physical Address Extensions, rozšírenie fyzického adresného priestoru na 36-bit.

Prostredie - Nastavenie procesora, ovplyvňuje význam inštrukcií, či budú pracovať so 16 alebo 32-bitovými dátami.

Režim - Stav procesora, ktorý určuje celkové správanie pri spracovávaní inštrukcií. Ovplyvňuje najmä prostredie, adresáciu, systém ochrany a obsluhu prerušení.

Segment - Časť adresy, popisuje úsek v pamäti. Získava sa z neho bázová adresa.

Segmentový register - Miesto v procesore (premenná), kam sa ukladá selektor segmentu, s ktorým chceme pracovať.

Selektor - 16-bitové číslo, ktoré sa skladá z indexu do tabuľky deskriptorov, typu tabuľky (GDT alebo LDT) a prístupových práv (RPL). Vyberá konkrétny segment z tabuľky.

Stránka - Page, časť fyzickej pamäte, najmenšia pamäťová jednotka, ktorej sa dá samostatne nastaviť prístupové práva, mapovanie do lineárneho adresného priestoru. Veľkosť 4 kB, 2 MB alebo 4 MB podľa požiadaviek OS.

Tabuľka deskriptorov - viď GDT/LDT.

Tabuľka stránok - Štruktúra obsahujúca niekoľko ukazovateľov na stránky v takom poradí, ako sú prístupné v lineárnom adresnom priestore.

Tenký klient - Jednoúčelové zariadenie, terminál, ktorý sa dokáže pripojiť na centrálny počítač. Pracuje sa s ním ako s bežným PC, ale výpočty a programy sa vykonávajú na vzdialenom centrálnom PC. Pre nízke nároky na výpočtovú silu tenkého klienta je možné použiť lacnú hardwarovú výbavu. Anglicky: Thin client.

TSS - Segment na uloženie stavu úlohy (angl. Task State Segment). Je to typ systémového deskriptora. Slúži na uloženie stavu procesora (všetky registre, návratová adresa, pár vecí naviac) pri zmene úlohy. Priamo podporuje multitasking - každá úloha (proces) môže mať svoj TSS a procesor medzi nimi môže prepínať.

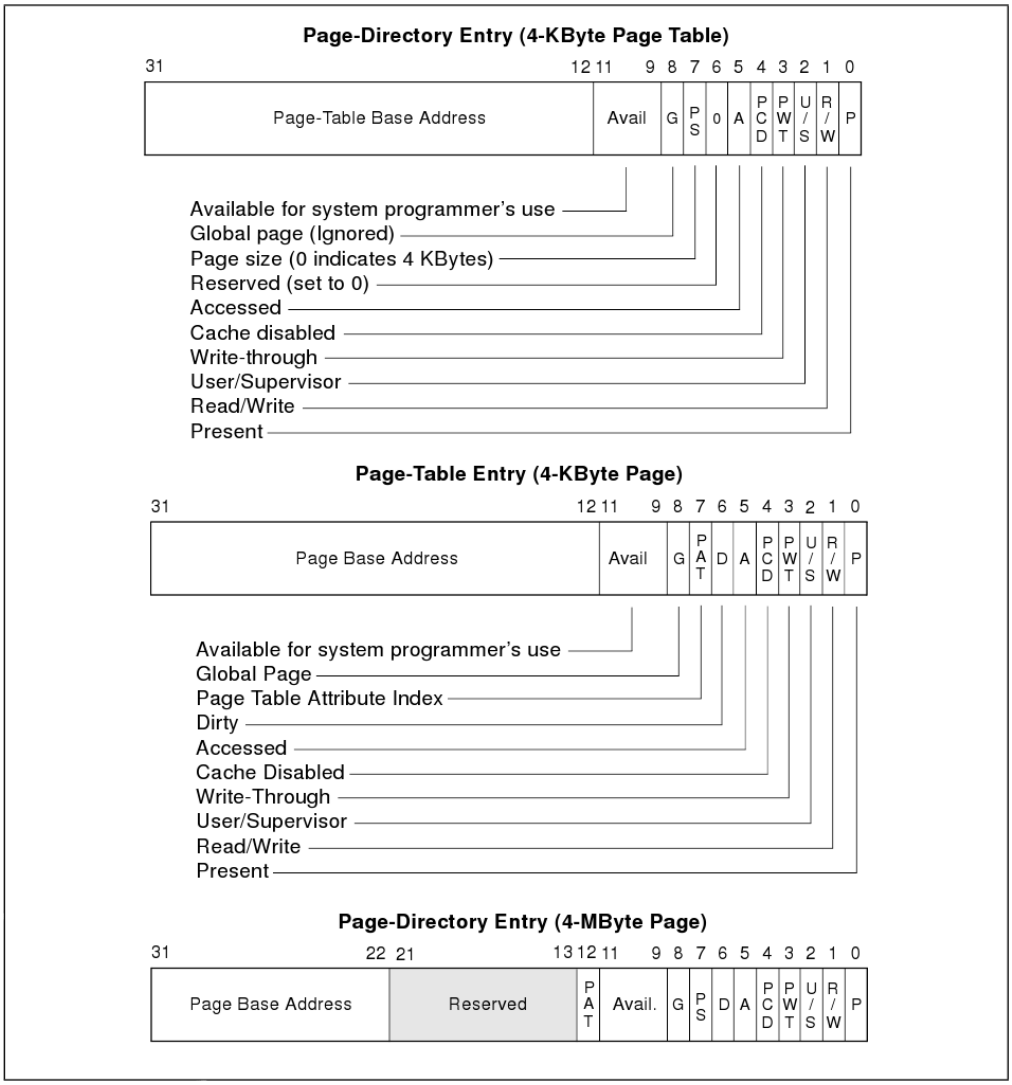
Virtuálny stroj - Tiež virtuálne PC. Prostredie pre beh softwaru určeného pre danú platformu. Anglicky: Virtual machine.

VMM - Správca virtuálneho stroja. Tiež Hypervisor. Dôležitá súčasť virtuálnych strojov, vytvára prostredie pre hosťovské OS, prideľuje virtuálne zariadenia, koordinuje prístup ku fyzickym zdrojom. Anglicky: Virtual machine monitor.

Volacia brána - Typ systémového deskriptora. Obsahuje selektor kódového segmentu a dá sa priradiť do kódového segmentového registra CS. Pri priradení (cez CALL alebo JMP) sa priradí selektor, ktorý sa v bráne nachádza.

Dodatok

A Tabuľky stránok



Obr. 7: Formát záznamov v tabuľkách stránok

Význam jednotlivých polí:

Present - indikuje, či je daná stránka alebo tabuľka prítomná vo fyzickej pamäti.

Read/Write - práva na zápis. 0 znamená len na čítanie, 1 povoľuje zápis.

User/Supervisor - práva ku stránke. 0=supervisor, 1=user.

Write Through - vyberá spôsob cachovania stránky.

Cache Disabled - ovláda, či bude stránka cachovaná.

Accessed - indikuje, že zo stránky sa čítalo.

Dirty - indikuje, že do stránky boli zapísané dáta.

Page Size - prepína veľkosť stránok.

Page Table Attribute Index - volí atribúty stránky.

Global - stránka je globálna a nepodlieha čisteniu TLB.

Available - k dispozícií pre programátora.

B Inštalácia a CD médium

Súčasťou práce je mini CD, ktoré obsahuje všetky zdrojové súbory programu Prot386 vrátane súboru Makefile na skompilovanie. Nachádza sa tam tiež skompilovaná verzia pre prípad, že nemáte možnosť zdrojové súbory skompilovať. CD je bootovateľné a program sa z neho naštartuje ak ho necháte pri spustení počítača v mechanike. (Treba mať v BIOS-e povolené bootovanie z CD).

Pre úspešné skompilovanie je potrebný gnu gcc kompilátor, gnu assembler a balíček bin86 obsahujúci programy as86 a ld86. Postup je nasledovný:

```
make clean && make
```

Výsledkom bude binárny súbor Image, ktorý sa dá zapísať na floppy disketu alebo USB kľúč ako raw image. Pod systémom Windows je potrebné k takémuto zápisu využiť utilitu rawrite.exe. Pod UNIX-ovým systémom použijeme *dd*:

```
dd if=Image of=/dev/sdb
```

Pritom */dev/sdb* nahradíme správnym menom zariadenia. Ak chceme vytvoriť CD médium, je nutné vybrať si možnosť "bootable CD", vybrať voľbu "emulate floppy disk" a zadať cestu ku Image súboru.

Literatúra

[1] Ambrož, P.: *Práca procesorov i386 v chránenom režime*,
Bakalárska práca, FMFI-UK, 2007

[2] Vagner, L.: *AThelp 1.50 Elektronický manuál*,
1996

[3] Brandejs, M.: *Mikroprocesory Intel 8086 - 80486*,
Grada, 1991

[4] Intel Corporation: *Intel 64 and IA-32 Architectures Software Developer's Manual Volume 1: Basic Architecture*,
2007

[5] Intel Corporation: *Intel 64 and IA-32 Architectures Software Developer's Manual Volume 2: Instruction Set Reference*,
2007

[6] Intel Corporation: *Intel 64 and IA-32 Architectures Software Developer's Manual Volume 3: System Programming Guide*,
2007

[7] Mann, Andi: *Virtualization 101*,
Enterprise Management Associates (EMA)

[8] Jason Nieh, Ozgur Can Leonard: *Examining VMware*,
Dr. Dobb's Journal August 2000

[9] John Scott Robin, Cynthia E. Irvine: *Analysis of the Intel Pentium's Ability to Support a Secure Virtual Machine Monitor*,
Proceedings of the 9th USENIX Security Symposium
Denver, Colorado, USA, 2000

[10] Popek, Gerald J.; Goldberg, Robert P.: *Formal Requirements for Virtualizable Third Generation Architectures*,
Communications of the ACM 17, 1974

[11] Lawton, Kevin P.: *Running multiple operating systems concurrently on an IA32 PC using virtualization techniques*,
2000

Zdrojové kódy programu Prot386 boli inšpirované jadrom Linux 0.01, ktoré je k dispozícií na http://www.kernel.org/pub/linux/kernel/Historic/