FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY
UNIVERZITY KOMENSKÉHO V BRATISLAVE

Katedra informatiky

Práca procesorov i386 v chránenom režime

Bakalárska práca

Bratislava 2007 Peter Ambrož

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

Práca procesorov i386 v chránenom režime

Bakalárska práca

Peter Ambrož
Vedúci bakalárskej práce: RNDr. Jaroslav Janáček
Bratislava 2007

Čestné prehlásenie

Čestne prehlasujem, že túto bakalársku 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 bakalárskej práce je popísať fungovanie najpoužívanejších desktopových procesorov spolu s príkladmi využitia jednotlivých funkcionalít. Sústredíme sa najmä na techniky využité v operačných systémoch.
Dôležitou súčasťou práce je spustiteľný program, ktorý demonštruje vybrané procesy interaktívnym spôsobom. Cieľová skupina sú najmä začínajúci systémoví programátori.

Kľúčové slová: Chránený režim, systémové programovanie

Abstract

Topic of this bachelor's thesis is to describe working of the most popular desktop processors also with some examples. 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 are beginner or intermediate system programmers.

Key words: Protected mode, system programming

Obsah

1 Úvod

Rozvoj počítačov ide míľovými krokmi vpred. Vyvíjajú sa nové technológie, či už po stránke hardwarovej alebo softwarovej. Srdcom všetkých počítačov je procesor, ktorý riadi ostatné zariadenia. Na poli domácich počítačov jednoznačne vedú procesory Intel Architecture (hoci výrobcov procesorov tejto architektúry je oveľa viacej, práve firma Intel priniesla na trh prvý takýto procesor).

My sa budeme zaoberať princípmi, ktoré fungujú v procesoroch Intel 80386 alebo vyšších (ďalej len i386). Model i386 som si nevybral náhodou. Je to práve pre množstvo noviniek a vylepšení, ktoré so sebou priniesol (viac v sekcií 2). Je to tiež z dôvodu veľkej podobnosti k tým dnešným[1] procesorom. Od čias i386 pribudlo množstvo inštrukcií, zlepšila sa rýchlosť ich spracovania, no k zásadnému zlomu došlo iba nedávno prechodom na 64-bitovú architektúru. Tieto procesory však vedia pracovať aj v režime kompatibilnom s 32-bit.

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.

Budúcim systémovým programátorom poslúži táto práca ako úvod do problematiky písania programov na spomínanej platforme. Priložený program umožní záujemcom odskúšať si niektoré aspekty správania sa procesora. Zdrojový kód poslúži ako základ pre písanie ďalších aplikácií, napríklad ako základ operačného systému.

[1] Pod pojmom "dnešný" sa rozumie aktuálna ponuka 64-bitových procesorov v roku 2007

2 História

História procesorov rodiny Intel pozostáva z viacerých viac či menej úspešných modelov, ktoré si teraz zhrnieme.

- **8086** bol prvý úspešný 16-bitový procesor. Vedel obslúžiť 1 MB pamäte a pracoval v jedinom režime, ktorému sa hovorí reálny mód[2]. Spolu s 8088 sa stali základom PC. Do tejto kategórie spadajú aj modely V20, V30, V40, V50, 80186, 80188. Všetky bežali na frekvencii 4.77 MHz, čo by sa v dnešnej dobe hodilo možno na hranie piškvoriek. Vo svojej dobe však neboli žravé systémy ako MS Windows a preto to nikomu neprekážalo.

- **80286** trocha vylepšený 8086 sa pomerne dlho udržal na výslní. Priniesol základ chráneného režimu[3]. Táto novinka sa takmer vôbec neujala, lebo programy písané pre 8086 nemohli bežať v chránenom režime. Síce sa tým vývojári ochudobnili o možnosť pristupovať ku 16 MB pamäti, toto však nikoho veľmi nebolelo, lebo priemerná 286-tka mala práve 1 MB RAM.

- **80386** plne 32-bitový procesor sa vďaka MS Windows stal nutnou súčasťou každého počítača. Procesory 286 a horšie signalizovali, že sa nejedná o počítač, skôr o počítadlo. 386 prináša plnohodnotný chránený režim, taký ako ho poznajú dnešné 32-bitové procesory, stránkovanie[4], možnosť adresovať 4 GB pamäte a tiež Virtual-8086 režim, ktorý slúži na spúšťanie aplikácií, pôvodne napísaných pre reálny mód procesorov 8086/286, pod chráneným režimom.[5] Pre kompatibilitu je možné používať aj chránený režim tak ako fungoval na 80286. Procesorom 80386 vďačia programátori tiež za nočné mory pri rozlišovaní 16 a 32 bitového kódu a za príjemné spomienky na staré 8-bitové Atari odložené na povali, kde bolo všetko také jednoduché. Varianty: 80386SX (bez koprocesora), 80386DX.

- **80486** priniesol najmä 8 KB internej cache. Sem spadajú modely 80486SX, 80486DX, 80486DX/2, 80486DX/4. Taktovacia frekvencia zbernice je u všetkých 25 MHz. Číslo za lomítkom uvádza násobok tejto frekvencie, na ktorej beží procesor. Posledný uvedený teda dosahoval taktovaciu frekvenciu až 100 MHz. Bol to skôr obchodný ťah na zákaznú-

[2] angl. Real mode
[3] angl. Protected mode
[4] angl. Paging
[5] Toto chýbalo chránenému režimu na 286-tke a bránilo jeho rozšíreniu sa

kov, lebo 4-násobná frekvencia procesora neznamenala 4-násobný výkon. Najmä kvôli čakaniu na pomalšiu zbernicu a pamäte bol reálny výkon maximálne 3-násobný. Variant 80486SX nemá koprocesor, resp. je to 80486DX s hardwarovo odstaveným koprocesorom.

- **Pentium** dokáže spracovať za vhodných podmienok 2 inštrukcie naraz. Lepšia virtualizácia procesov vo V86 režime, nové možnosti stránkovania, SMM mód. Zbernica dosahuje rýchlosti 50-66 MHz, procesory 75-200 MHz.

- **P6** priniesol zlepšenie superskalárnej architektúry. Od 3 do 8 inštrukcií spracovateľných v jednom hodinovom takte. Do tejto kategórie patria procesory Pentium Pro, Pentium II, Pentium II Xeon, Celeron, Pentium III a Pentium III Xeon

- **Pentium 4** v novších modeloch už ponúka aj 64-bit architektúru, podporu pre hardwarovú virtualizáciu, ktorá bola ďalej vylepšená v procesoroch radu **Xeon**.

Pri každom modeli všeobecne platí, že je rýchlejší, ako jeho predchodca a má niekoľko nových inštrukcií.

3 Základné princípy

Pre ďalší výklad je potrebné oboznámiť čitateľa so základnými princípmi, aby v texte nevznikali problémy s pochopením.

Procesor od svojho zapnutia neustále vykonáva inštrukcie, ktoré má uložené v operačnej pamäti. Inštrukcie sú uložené ako postupnosti bytov, ktorým procesor rozumie, dokáže ich dekódovať a vykonať. K dispozícií sú inštrukcie na počítanie (aritmetické), skákanie v programe (riadiace), prácu s I/O zariadeniami, presuny dát v pamäti, ovplyvňovanie vnútorného stavu procesora (zápis do kontrolných registrov, atď...).

3.1 Prostredie 16/32

32-bitové procesory umožňujú pracovať s dátami o veľkosti 8, 16 a 32 bitov. Inštrukcie pre prácu s 8 a 16-bitovými dátami majú odlišný zápis, preto nevznikajú žiadne problémy s ich rozlišovaním. Inštrukcie pre prácu s 32-bitovými dátami majú rovnaký zápis ako tie 16-bitové (po preklade do strojového kódu sa zapíšu rovnakými bajtmi). Význam takejto inštrukcie sa určuje práve pomocou prostredia.

Existujú 2 prostredia: 16-bitové a 32-bitové. To, ktoré je aktuálne vybraté, určuje význam 16/32 inštrukcií. Veľkosť prostredia je buď pevne určená režimom práce, alebo sa dá nastaviť pre jednotlivé časti programu. V oboch prípadoch sa dajú použiť ešte aj inštrukčné prefixy na lokálnu zmenu prostredia pre nasledujúcu inštrukciu. Inštrukčné prefixy obracajú význam aktuálneho prostredia zo 16 na 32 a naopak. Význam prefixov je len v kombinácií so 16/32 inštrukciami a na 8-bitové inštrukcie nemajú žiaden vplyv.

3.2 Adresácia

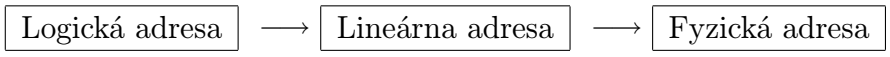
Práci s pamäťou sa hovorí adresácia. Pamäť je z pohľadu procesora jedno obrovské pole bytov, kde každý uložený byte má svoju presnú pozíciu - adresu. Triviálna predstava nám hovorí, že keď uvedieme v inštrukcií nejakú adresu, tak táto adresa bude vlastne indexom do poľa - do pamäte. V skutočnosti však procesor so zadanou adresou (budeme jej odteraz hovoriť logická adresa) prevádza niekoľko výpočtov, až nakoniec dostane ozajstné poradové číslo - fyzickú adresu.

Prvý výpočet sa nazýva segmentácia a jeho cieľom je transformovať logickú adresu na lineárnu, čo je akýsi medzivýsledok pre ďalšie výpočty. Pamäť sa logicky člení na segmenty - úseky, v rámci ktorých sa vyberá adresa pomocou offsetu. Logická adresa sa teda skladá dvoch častí: Segment a offset. Pomocou segmentácie ju prevádzame na lineárnu adresu, ktorá je v podobe

jediného čísla. Presný prepočet bude popísaný v sekciách 4.1 a 4.2. Lineárna adresa sa zvykne rovnať fyzickej adrese. Ale len v prípade, že nie je v činnosti nasledujúci krok - stránkovanie.

Stránkovanie je proces na prevádzanie lineárnych adries na fyzické adresy. Tento proces nemusí byť vždy v činnosti. Slúži na virtualizáciu pamäte. Stránkovanie vníma celú pamäť v 4 kB alebo 4 MB veľkých blokoch[6], nazývaných stránky, ktoré je možné ľubovoľne preusporiadať. Nie každá stránka musí byť mapovaná do fyzickej pamäte - tu vzniká priestor pre odkladanie stránok na disk (tzv. swapping). Z nesúvislých oblastí fyzickej pamäte je možné urobiť súvislé bloky adries, operačný systém môže jednotlivým procesom odoprieť prístup ku cudzej pamäti, atď.... Výstupom je fyzická adresa, ktorú procesor použije pri prístupe do pamäte. Pokiaľ je stránkovanie vypnuté, fyzická adresa = lineárna adresa. Proces stránkovania je transparentný, užívateľ vníma iba lineárne adresy. Ozajstné fyzické adresy na zbernici pozná len procesor[7].

Celý postup pri adresácií zachytáva obr. 1.



Obr. 1: Postup pri adresácií.

[6] 4 MB stránky prináša až pentium

[7] a tiež ten, kto implementoval stránkovanie v danom systéme

4 Režimy práce

Niekoľkokrát tu už boli spomenuté akési režimy. Teraz si ozrejmíme, čo to znamená.

Režim práce[8] (alebo tiež mód) je stav, v ktorom sa procesor môže nachádzať. 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**. S príchodom 64-bitovej architektúry prichádzajú ďalšie 2 módy: **IA-32e Kompatibilný** a **IA-32e 64-bit**. To už je celkom slušná zbierka režimov, takže si ich teraz postupne popíšeme.

4.1 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. Presne tak, ako tomu bolo aj na procesoroch 8086.[9]

Reálna adresácia používa logické adresy v tvare segment:offset. Segment a offset sú vždy 16-bitové čísla.[10] Segment priamo určuje bázovú adresu, ktorá sa vypočíta zo vzorca:

*base = 16 * segment*

Ku bázovej adrese sa potom pripočíta offset, čím dostaneme lineárnu adresu.

*linearaddress = 16 * segment + offset*

Stránkovanie je v reálnom móde vypnuté, takže lineárna adresa = fyzická adresa.

Takýmto spôsobom sa dá adresovať niečo málo nad 1 MB, presnejšie 1114096 B. Pozornému čitateľovi isto neunikol fakt, že segmenty sa po 16 bytoch prekrývajú, takže napr. adresa 0x3F00:0x0001 je tá istá ako 0x3000:0xF001. Stačí si zrátať fyzické adresy, v oboch prípadoch vyjde tá istá - 0x3F001. Najvyššia logická adresa je 0xFFFF:0xFFFF $\Rightarrow$ fyzická adresa = 0x10FFEF = 1114095. Procesory 8086 mali iba 20-bitovú adresnú zbernicu, takže viac ako 1048576 B pamäte nemohli dosiahnuť, hoci výpočet to dovoľuje. Adresy nad 1 MB sa cyklicky premietali na začiatok adresného priestoru.

V reálnom móde majú aplikácie dovolené všetko. Prístup ku pamäti a I/O zariadeniam je neobmedzený. V reálnom móde pracuje operačný systém MS-

[8] angl. Operation mode

[9] 8086 síce nemalo žiadne privilegované inštrukcie, to však nášmu tvrdeniu neodporuje

[10] Ak by sme zadali 32-bitový offset, procesor by použil len spodných 16 bitov.

DOS a obrovské množstvo dnes už zabudnutého softwaru ale aj nesmrteľných hier, ktoré v rokoch 80-tych a 90-tych na konci 20-teho storočia úplne ovládli domáce počítače. Nadšenci týchto dôb môžu dodnes využívať reálny mód svojich Intel Architecture procesorov na občasné zaspomínanie si. V praxi sa reálny mód používa už len niekoľko málo sekúnd po zapnutí počítača až po štart operačného systému, keďže tie dnešné OS už pracujú výhradne v režime chránenom a využívajú tak všetky výhody tohto prostredia.

4.2 Chránený mód

Chránený mód je natívny mód, v ktorom je možné využiť naplno všetku funkcionalitu procesora a adresovať celú dostupnú pamäť. Vstupuje sa doň prepnutím z reálneho módu (viac v sekcií 6.1.1). Prostredie je 32-bitové, ale dá sa pre každý segment zvlášť nastaviť, či bude 32-bit alebo 16-bit. 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.

Chránená adresácia používa logické adresy v tvare selektor:offset. Offset je 32-bitový (resp. toľko, aká je aktuálna veľkosť prostredia). Na rozdiel od reálneho módu, bázová adresa sa už nepočíta priamo zo segmentu (aj preto sa mu už nevraví segment, ale selektor). Selektor je 16-bitové číslo a obsahuje index do tabuľky deskriptorov.[11] Deskriptor popisuje vlastnosti jedného segmentu. V deskriptore je uložená bázová adresa segmentu (32-bit), limit - najvyšší povolený offset v danom segmente, typ segmentu - systémový, kódový alebo dátový, veľkosť prostredia, prístupové práva (DPL) a iné užitočné informácie. Tabuľky deskriptorov existujú 3: Globálna (GDT), lokálna (LDT), tabuľka prerušení (IDT). Pre prístup do pamäte je možné využiť iba deskriptory uložené v GDT alebo LDT. Každá z týchto tabuliek môže obsahovať až 8192 deskriptorov. Selektor obsahuje okrem indexu ešte aj informáciu o použitej tabuľke (GDT alebo LDT) a tiež prístupové práva (RPL), ktorými sa budeme zaoberať neskôr.

Ako to teda funguje? Lineárna adresa sa získa pripočítaním offsetu ku bázovej adrese. Odlišnosť oproti reálnemu módu je v spôsobe získania bázovej adresy. Táto je uvedená v deskriptore segmentu, ktorý sme si vybrali indexom do tabuľky deskriptorov. Index a indikátor tabuľky sú uložené v selektore, ktorý je súčasťou logickej adresy. Dochádza tu teda ku nepriamej adresácií, keďže už bázovú adresu nevieme vybrať priamo číslom segmentu, avšak vyberáme si segment z tabuľky deskriptorov, ktorý má vopred nastavenú bá-

[11] Slovíčka "selektor" a "deskriptor" je vhodné pochopiť čím skôr, lebo sa budú často využívať v ďalšom texte

zovú adresu. Toto dáva operačnému systému možnosť zabrániť v prístupe ku ľubovoľnej lineárnej adrese. Pamäť je prístupná len cez deskriptory, a ich bázové adresy a limity sú predom nastavené na vhodné hodnoty. Užívateľský proces nemá šancu zmeniť tieto hodnoty, pokiaľ je systém ochrany správne nastavený (stačí, že tabuľky GDT a LDT nebudú uložené na adresách dostupných cez deskriptory, ktoré sa v nich nachádzajú).

linearaddress = GDT[selector.index].base + offset

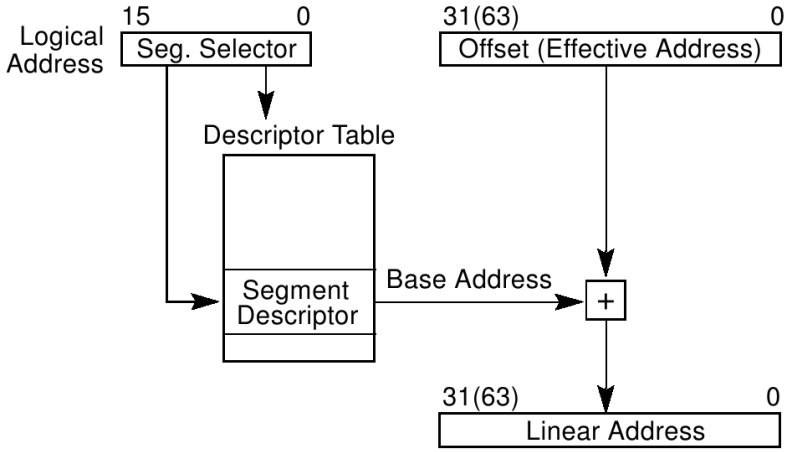
resp.

linearaddress = LDT[selector.index].base + offset

Stránkovanie môže byť zapnuté a lineárne adresy sa tak môžu prepočítavať na fyzické adresy.

Index (13)	TI (1)	RPL (2)

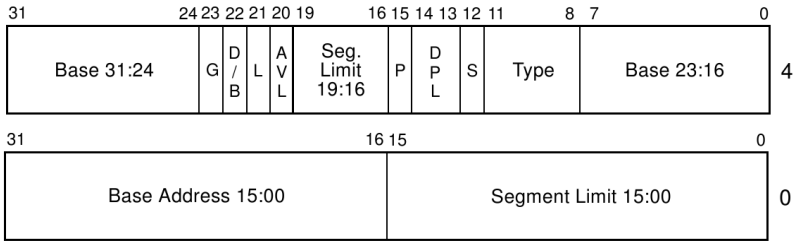
Obr. 2: Štruktúra selektoru: Index do tabuľky, TI - Table indicator - 0 = GDT, 1 = LDT. Počet bitov je uvedený v zátvorke.



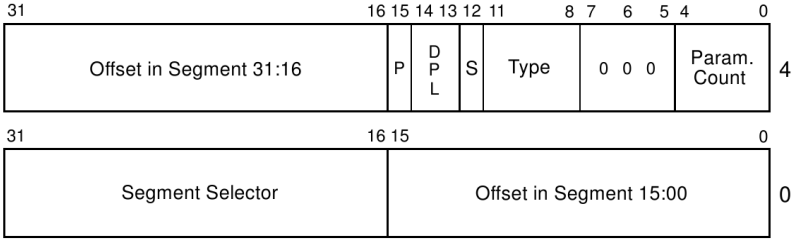
Obr. 3: Schéma chránenej adresácie

Princíp chráneného módu tkvie v ochrane pamäte medzi operačným systémom a programami za pomoci segmentácie a stránkovania. Snaha je, aby užívateľský program nemohol nijako ohroziť stabilitu systému, keďže sa počíta s behom viacerých programov naraz (multitasking). V prípade dobre napísaného operačného systému je možné takúto ochranu dosiahnuť.

Fyzická adresa začiatku tabuľky GDT v pamäti je uložená v registri GDTR. Zápis do registra GDTR prebieha jedine pomocou inštrukcie LGDT, ktorá je privilegovaná - dá sa použiť iba pri CPL = 0.



Obr. 4: Štruktúra deskriptoru



Obr. 5: Štruktúra deskriptoru brány

4.2.1 Štruktúra deskriptoru

Limit 15:0 - bity 0-15 obsahujú spodných 16 bitov limitu daného segmentu.

Base 23:0 - bity 16-39, spodných 24 bitov bázovej adresy.

Type - bity 40-43, typ deskriptora

S - bit 44, 0 = systémový, 1 = kód alebo dáta. Bity S a Type spoločne určujú výsledný typ deskriptora.

DPL - bity 45-46, úroveň oprávnenia deskriptora.

P - bit 47, present. Prítomný = 1, neprítomný = 0.

Limit 19:16 - bity 48-51, horné 4 bity limitu. Dokopy 20 bitov.

AVL - bit 52, available. Môže ho využiť OS pre svoje účely.

L - bit 53, iba pri kódových segmentoch, prepína medzi 32-bit kompatibilným a 64-bit.

D/B - bit 54, iba pri nesystémových deskriptoroch. Veľkosť prostredia, 0 = 16-bit, 1 = 32-bit.

G - bit 55, granularita segmentu. Ak G = 1, tak limit sa násobí 4096 (s doplnením 1-tiek sprava). Takto je možné získať limit až veľkosti 4 GB. Takémuto segmentu sa potom hovorí "flat", lebo pokrýva celý adresný priestor.

Base 31:24 - bity 56-63, horných 8 bitov bázovej adresy. Dokopy 32 bitov.

Offset 15:0 - bity 0-15, spodných 16 bitov offsetu od začiatku segmentu.

Selector - bity 16-31, selektor segmentu, ktorý je bránou nepriamo volaný.

Param.Count - bity 32-36, počet parametrov (double-word), ktoré sa skopírujú zo starého zásobníka na nový v prípade že dôjde ku zmene zásobníkov z dôvodu zmeny CPL.

4.3 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 ozajstny reálny mód. Toto všetko sa však deje vo vnútri chráneného módu a každý "prehrešok" V86 procesu voči systému ochrany vyvoláva výnimku. Obsluha týchto výnimiek môže emulovať prostredie, ktoré bežiaci proces očakáva. Napr. proces niečo zapíše na I/O port. Toto v chránenom režime vyvolá výnimku. Obsluha výnimiek pre V86 mód môže buď zápis vykonať alebo sa tak aspoň zatvárať a vrátiť riadenie procesu, ktorý sa o výnimke nedozvie.

Bežať takto môže viac procesov. Keďže každý z nich využíva reálnu adresáciu, je nutné použiť stránkovanie na oddelenie ich adresných priestorov. Všimnime si výhodu stránkovania - všetky V86 procesy si myslia, že používajú lineárne adresy z rozsahu 0-1 MB. Stránkovanie však zabezpečí, že fyzické adresy sa neprekrývajú.

Virtual-8086 mód sa využíva napr. vo Windows na spúšťanie MS-DOS aplikácií v okne. MS-DOS aplikácie nedokážu bežať v 32-bit chránenom režime, ale V86 im často stačí. Kto zvykol používať pod MS-DOS rôzne memory managery ako napr. EMM386 alebo QEMM, bol vlastne vo V86 móde a nie v reálnom ako si možno myslel. Pamäť nad 1 MB sa nazývala rozšírená (extended príp. expanded memory). EMM386 využíva stránkovanie na dosiahnutie pamäte nad 1 MB a toto by v reálnom móde nebolo možné. Pomocou stránkovania sa dá premapovať stránky z rozšírenej pamäte dolu do oblasti 0-1 MB, ktorá je dosiahnuteľná procesmi v reálnom / V86 móde.

Virtual-8086 mód sa dá zapnúť len z chráneného módu. V reálnom móde nemá opodstatnenie a pokus o jeho zapnutie je ignorovaný.

4.4 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. Do SMM módu a von sa dá prepnúť z ľubovoľného iného módu. Prepnutie sa však dá vykonať iba hardwarovo - privedením impulzu na SMI pin procesora alebo pomocou prerušenia od systému APIC. Pri vstupe do SMM procesor uloží stav aktuálneho výpočtu. Opustenie SMM módu a pokračovanie v prerušenej činnosti sa robí softwarovo. Typickým príkladom využitia SMM je uspanie notebooku alebo "upratanie" RAM po zatvorení veka. V praxi sa tento mód využíva zriedka práve kvôli väzbe na HW.

4.5 64-bitové módy

Tieto režimy spomíname iba pre kompletnosť informácie, keďže sa venujeme 32-bitovým procesorom.

4.5.1 IA-32e Kompatibilný

Prostredie pre spúšťanie programov písaných pre 32-bitový chránený režim. Väčšina programov pobeží. Výhoda oproti použitiu pôvodného chráneného režimu je v tom, že IA-32e kompatibilný režim je možné spustiť priamo pod 64-bitovým režimom. Je to akási analógia virtual-8086 režimu, ktorý sa púšťa pod chráneným režimom.

4.5.2 IA-32e 64-bit

Plnohodnotné 64-bitové prostredie a možnosť adresovať 64-bit pamäte. Programy písané pre 32-bit nebudú bežať v tomto móde.

5 Systém ochrany

Systém ochrany je neoddeliteľnou súčasťou chráneného módu procesora a pomáha systémovému programátorovi dosiahnuť stabilitu OS. V prípade jediného bežiaceho procesu v reálnom móde toto nebolo potrebné. Akonáhle chceme povoliť viac procesov bežiacich naraz, je nutné chrániť dôležité zdroje - I/O zariadenia a pamäť pred neželaným alebo náhodným prístupom.

Systém ochrany nie je "dokonalý super anti-hacker systém", ale je to sada pravidiel a oprávnení, ktoré pomáhajú udržať poriadok. V prípade správne nastaveného systému ochrany je nemožné, aby aplikácia ohrozila beh OS.

5.1 Úrovne oprávnenia

Procesor rozoznáva 4 úrovne oprávnenosti vyjadrené číslom od 0 po 3. Aktuálna úroveň sa nazýva CPL (Current Privilege Level). Toto má každý bežiaci proces uložené a nedokáže to priamo meniť.

- CPL = 0: Najvyššie právomoci. Určené pre jadro operačného systému a ovládače zariadení. Dovolené je všetko (či už priamo alebo nepriamo).
- CPL = 3: Najnižšie práva. Všetky užívateľské aplikácie.

Medzistupne 1 a 2 môžu byť použité pre jemnejšie škálovanie prístupových práv, v praxi sa však takmer vôbec nepoužívajú. Ekvivalentná ochrana sa dá dosiahnuť aj s pomocou len 2 úrovní 0 a 3.

Každý proces má uložený selektor segmentu, ktorý práve vykonáva, v registri CS. Hodnota CPL je uložená práve v poli RPL tohto selektora. Pojem CS.RPL je teda ekvivalentný s CPL. Pri udeľovaní prístupu sa často porovnáva CPL s nejakou inou hodnotou a to:

DPL - Descriptor Privilege Level - prístupové práva deskriptora, uložené v každom deskriptore. Kontroluje sa pri prístupe ku dátovým a systémovým segmentom. Prístup je povolený, ak CPL $\leq$ DPL. Inak povedané, proces musí mať práva aspoň také alebo vyššie ako deskriptor.[12]

RPL - Requested Privilege Level - dodatočné práva, uložené v každom selektore. Dá sa nastaviť vždy pri plnení segmentového registra (okrem CS). Slúži pre dodatočnú kontrolu prístupu ku dátovým segmentom, pretože RPL $\leq$ DPL musí platiť pre úspešný prístup. Špeciálny význam má pole RPL v selektore aktuálneho kódového segmentu (uložený v CS), keďže toto určuje CPL.

[12] Pozor na nesúlad: vyššie práva znamenajú numericky nižšie CPL a naopak.

IOPL - Input Output Privilege Level - práva na prácu s I/O zariadeniami. Uložené v registri EFLAGS. Pokiaľ CPL $\leq$ IOPL, proces môže robit vstup a výstup na I/O porty a tiež používať inštrukcie CLI, STI. V opačnom prípade je povolené pracovať iba s portami, ktoré sú vymenované v neformátovanej časti TSS.

5.2 Pravidlá

Systém ochrany má isté pravidlá, ktoré uplatňuje v rôznych situáciách. V prípade nedodržania niektorého z nich, procesor vygeneruje výnimku, ktorú následne spracuje ako prerušenie. Najčastejšie je generovaná výnimka 0x0D - General Protection Fault (Všeobecná chyba ochrany). V prípade inej výnimky je číslo prerušenia uvedené na konci pravidla v hexadecimálnom zápise. (Napr. INT 0C znamená že pri nedodržaní danej podmienky dochádza ku prerušeniu 0x0C - Chyba zásobníka). Pravidlá systému ochrany:

1. Programy môžu pracovať iba s pamäťou dostupnou cez deskriptory, ak platí CPL $\leq$ DPL daného deskriptora. Inak INT 0D.

2. Programy môžu pracovať s I/O zariadeniami, ktoré majú dovolené v poli TSS, alebo pokiaľ CPL $\leq$ IOPL.

3. Zmena príznaku IF v registri EFLAGS je povolená iba ak CPL $\leq$ IOPL.

4. Každá úroveň oprávnenosti má svoj vlastný zásobník[13] (jeho selektor je v registri SS). Musí platiť SS.RPL = CPL = DPL (deskriptor segmentu zásobníka).

5. Program nemôže zapisovať do segmentu uloženého v CS.

6. Z kódového segmentu sa dá čítať iba ak má nastavený bit R vo svojom deskriptore.

7. Z dátového segmentu sa dá čítať vždy. Zapisovať do dátového segmentu možno iba ak má nastavený bit W.

8. Priame priradenie hodnoty do CS (teda aj do CPL) nie je možné. INT 06 - Neprípustná inštrukcia.

9. Program nesmie prekročiť limit príslušného segmentu s ktorým pracuje. Offset $\leq$ limit.

[13] angl. Stack

10. Program nesmie podtiecť so zásobníkom. Ak počas práce so zásobníkom nastane ESP < 0, generuje sa INT 0C.

11. Program s CPL > 0 nesmie vykonávať privilegované inštrukcie.

12. Program môže meniť CS nepriamo pomocou inštrukcií CALL, JMP, RET (tiež INT, IRET).

13. Program môže pomocou CALL alebo JMP zmeniť CS len pokiaľ CPL = DPL nového segmentu. CPL sa nezmení.

14. Program môže pomocou CALL alebo JMP zmeniť CS aj v prípade CPL $\geq$ DPL, pokiaľ je volaný segment conforming (C = 1). CPL sa nezmení.

15. Program môže znížiť svoje CPL (vyššie oprávnenia) nepriamo volaním cez bránu.

16. Program môže zvýšiť svoje CPL (nižšie oprávnenia) volaním RET, pokiaľ na zásobníku existuje štruktúra, akú vytvára volanie cez bránu.

17. Program môže použiť deskriptor, ktorý má nastavené P = 0 (neprítomný). Vyvolá sa INT 0B - výpadok segmentu a obsluha MUSÍ segment sprístupniť (inak sa zacyklí).

Pravidlá pri plnení segmentových registrov sú nasledovné:

5.2.1 Dátové segmenty

Registre DS, ES, FS, GS môžu byť naplnené iba selektorom dátového segmentu alebo selektorom kódového segmentu, pokiaľ je čitateľný (R = 1). Pred naplnením obsahu registra je vykonaný test prístupových práv. Splnená musí byť podmienka CPL $\leq$ DPL $\geq$ RPL. Hodnota DPL sa berie z deskriptora, na ktorý ukazuje selektor, ktorého hodnota je do registra plnená. RPL sa berie priamo zo selektora.

5.2.2 Segment zásobníka

Register SS môže byť naplnený iba selektorom dátového segmentu, ktorý naviac musí byť zapisovateľný (W = 1). Podmienka je CPL = DPL = RPL. Inak INT 0D.

5.2.3 Kódový segment

Plnenie CS sa nedá vykonať priamo priradením hodnoty, ale volaním podprogramu (CALL) resp. skokom (JMP). Rozdiel je iba v tom, že CALL ukladá na zásobník návratovú hodnotu, teda pôvodný selektor a offset, kde bol vykonaný skok. JMP neukladá nič. Parametrom volania musí byť selektor kódového segmentu, selektor TSS, volacej brány alebo TSS brány. Pri volaní selektora TSS alebo brány platí tá istá podmienka ako pri dátových segmentoch: CPL $\leq$ DPL $\geq$ RPL. Rozoznávame priame volanie alebo volanie cez bránu.

Priame volanie:
Parametrom je priamo kódový segment (jeho selektor). Záleží na príznaku "conforming". Ak C = 0, tak musí byť splnené: CPL = DPL, RPL $\leq$ CPL. Ak C = 1, tak postačuje CPL $\geq$ DPL, na RPL nezáleží. Pozor na fakt, že vzťah medzi CPL a DPL je opačný ako pri dátovom segmente. Po úspešnom volaní sa CS naplní vybraným selektorom. CPL procesu sa nezmení. Platí v prípade CALL aj JMP verzie.

Volanie cez bránu:
Volacia brána je druh deskriptora, ktorý obsahuje selektor iného kódového segmentu. Umožňuje lepšiu kontrolu nad právami použitia daného segmentu. Parametrom volania je selektor volacej brány. Najprv sa určí prístup ku segmentu brány (tak ako bežný dátový segment), prečíta sa cieľový selektor uložený v deskriptore brány a nakoniec sa vykoná kontrola práv na cieľový segment. Pokiaľ vykonávame JMP a cieľový segment je non-conforming (C = 0), musí platiť CPL = DPL cieľa. V ostatných prípadoch stačí CPL $\geq$ DPL cieľa. Naviac v prípade vykonávania CALL na bránu, ktorá obsahuje selektor non-conforming kódového segmentu, dôjde ku zmene CPL tak, že CPL:=DPL cieľa. Pokiaľ za zmení CPL (numericky sa zníži, práva sa zvýšia), zmení sa aj zásobník. Nové hodnoty SS a ESP sa získajú z aktuálneho TSS. Pôvodné hodnoty SS a ESP sa uložia na nový zásobník spolu s návratovou adresou CS a EIP. Do CS sa uloží selektor získaný z deskriptora brány (cieľový).

Brány sa používajú pri systémových volaniach OS. Aplikácia bežiaca pod CPL = 3 využíva služby OS, ktoré sa volajú cez bránu. Napr. s otvorením a zápisom do súboru sa aplikácia netrápi, namiesto toho zavolá systémové volanie. K takýmto operáciám treba CPL = 0, čo je presne prípad pre volaciu bránu, ktorá dokáže znížiť CPL počas volania na hodnotu volaného DPL. Pri návrate sa hodnota CPL upraví naspäť. OS dokáže regulovať prístup ku volacím bránam (kontroluje sa nielen prístup ku cieľovému segmentu ale aj ku

samotnej bráne). Keď už nejaké volanie sprístupní ako bránu, je to zvyčajne dobre otestovaný kód, ktorý neohrozí beh systému.

Návrat z volania:
Možný len pokiaľ sa použilo CALL. Prečíta sa návratová adresa zo zásobníka (selektor, offset) a uloží sa do registrov CS a EIP. Vykonávanie programu teda pokračuje za inštrukciou CALL, ktorá volanie spôsobila. Ak sa pri čítaní uloženého selektora zistí, že pri volaní sa zmenilo CPL (hodnota CPL uložená v selektore návratovej adresy je numericky vyššia ako aktuálna hodnota CPL - návrat do nižšej úrovne oprávnení), okrem CS a EIP sa obnovia zo zásobníka aj hodnoty SS, ESP, čiže selektor a offset pôvodného zásobníka.

Každý segmentový register s výnimkou SS môže byť naplnený aj tzv. nulovým selektorom (fyzicky číslo 0). V takom prípade je daný segmentový register neplatný a až pri prvom pokuse o prístup cezeň, procesor vyvolá výnimku.

Prístupové práva sa kontrolujú len pri priraďovaní selektora do segmentového registra. Ak táto operácia prejde, práva na segment sa pri ďalších prístupoch do segmentu nekontrolujú. Preto je potrebné, aby procesy pred návratom do nižšej úrovne oprávnení prepísali obsah segmentových registrov vhodnými hodnotami (napr. nulami). Hovorí sa tomu zneplatnenie (angl. Invalidate). Inak by mohol menej oprávnený proces naďalej pracovať so segmentmi, ktoré mu nepatria.

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 štúdovaním 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 i LDT. Užívateľ sa môže pohrať pri nastavovaní parametrov.
- Podpora viacerých LDT tak, ako to je aj v ozajstnom systéme. Je možné vyrobiť si viac LDT, naplniť ich deskriptormi a následne si prepínať 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.

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 prípadne UNIX a iné, keďže tieto bežia v chránenom móde a Prot386 by nemohol priamo pracovať s pamäťou a deskriptormi. Tento program funguje sám sebe ako operačný systém a spúšťa sa priamo pri štarte počítača z diskety, 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 stačí 640 kB RAM, pre plnú funkčnosť sú odporúčané 2 MB. Program sa nikam neinštaluje, vždy nanovo sa spúšťa priamo z média.

Veľká snaha bola kladená 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 budú popísané v sekciách 6.1.1 a 6.1.2.

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.[14] V našom prípade bol na začiatku média bootloader. Jeho úlohy sú jasné:

- Vypísať na obrazovku niečo v štýle "štartuje sa systém, prosím majte strpenie" - 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 sa pri bootovaní z CD alebo USB-kľúča vždy tvári, že toto zariadenie je klasická 1.44 MB disketa, čo nám nesmierne zjednodušuje celý kód.)

[14] alebo nejaký boot vírus, ktorý sa má postarať o sformátovanie pevného disku

6.1.1 Prepnutie do chráneného módu

Nasledujú úkony súvisiace s prepnutím do chráneného módu.

- Povoliť hradlo A20. Veľa ľudí tento krok prehliada a potom má netriviálne problémy v chránenom móde. Objasnenie viď nižšie.
- Premapovať hardwarové prerušenia z dôvodu menšej nezhody medzi IBM a Intel. Štandartne prerušnia od zariadení IRQ 0-7 vyvolávajú softwarové prerušenie INT 08-0F a to už od čias 8086 (žiaden chránený mód). Intel sa však rozhodol prideliť obsluhám kritických výnimiek sw. prerušenia INT 00-1F, čo sa prekrýva s pôvodným mapovaním IBM. V reálnom móde väčšina výnimiek nenastáva tak to nikomu neprekáža. Pri prepnutí do chráneného módu je nutné premapovať IRQ na vyššie INT-y. Robí sa to 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 GDT (privilegovaná inštrukcia LGDT).
- Nastaviť bit PE (Protection Enable) v registri CR0 (chránený mód zapnutý).
- 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 z diskety - 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, čo nám však jazyk assembler svojou veľkosťou produkovaného kódu nijako neprekazí.[15]

Všimnime si, že tabuľka GDT má kľúčové postavenie, lebo rozhoduje o prístupe ku lineárnym adresám a jej modifikácia užívateľským programom nie je želaná. Tomuto sa dá ľahko zabrániť. Vieme, že proces môže pracovať iba s lineárnymi adresami, ktoré má sprístupnené cez GDT alebo LDT. Stačí teda zabezpečiť, aby žiadny z deskriptorov v týchto tabuľkách nesprístupňoval lineárnu adresu GDT resp. LDT pre CPL > 0. Podobnú starostlivosť treba venovať aj tabuľkám LDT.

[15] V aktuálnej verzii bootloadera ostalo cca 28 B nevyužitých

6.1.2 Návrat do reálneho módu

Toto nieje úlohou bootloadera, no pre úplnosť si uvedieme postup, ako sa vrátiť do reálneho módu. Triviálny postup je reset počítača. Dá sa aj sofistikovanejšie:

- Vytvoriť deskriptory pre segmenty vhodné v reálnom móde - limit 64 kB
- Skočiť do takéhoto segmentu (naplniť ním CS)
- Naplniť aj ostatné segmentové registre vhodnými segmentami
- Zhodiť bit PE v registri CR0
- Vykonať skok na adresu reálneho módu (v tvare 16-bit segment:16-bit offset)

6.1.3 Hradlo A20

Je to jeden veľký strašiak z dávnych dôb, ktorý dodnes strháva systémových programátorov zo sna. Tu je jeho príbeh prevzatý z [1]:

Procesor 8086 mal 20 bitov adresnej zbernice, obslúžil teda 1048576 B. Zvyšných 65520 B adresovateľných v reálnom móde (spomeňte si na kapitolu 4.1) sa cyklicky premietalo späť. Takže adresy 0xFFFF:0x0000 a 0x0000:0x0010 alebo 0xFFFF:0x0001 a 0x0000:0x0011, atď ... sa premietali na tú istú fyzickú adresu.

S príchodom 80286 sa zbernica rozšírila a nebol viac dôvod, prečo adresy cyklicky premietať späť.

Niekto veľmi inteligentný sa domnieval, že existuje tak neprispôsobivý program pre 8086, ktorý adresuje segment 0x0000 pomocou segmentu 0xFFFF. A tak v záujme ešte vyššej kompatibility zaviedol hradlo A20. Je to bežné logické hradlo typu AND, umiestnené na 20. adresnej linke (čísluje sa od 0). Toto, pokiaľ je povolené, nerobí nič a necháva existovať horných 65520 B. Pokiaľ je zakázané, nuluje 20. adresnú linku procesora a efekt 8086 je na svete. Ovladanie A20 sa robí cez porty klávesnice (hradlo je zvyčajne pripojené na nevyužitú nožičku kontroléra klávesnice).

Pokiaľ program pracuje v chránenom móde, je nutné mať A20 povolené, inak každá operácia s pamäťou nad 1 MB vyvolá pravdepodobne[16] chybu.

[16] zaručene!

6.2 Obsluha výnimiek

Po úspechoch bootloadera sa začne vykonávať jadro systému, už v plne 32-bitovom chránenom prostredí. Segmentové regisatre sa naplnia selektormi, potrebné štruktúry sa inicializujú. Jednou z nich je aj tabuľka prerušení - IDT (Interrupt Descriptor Table).

Každá výnimka procesora a každé prerušenie od hardwaru (napríklad od klávesnice pri stlačení klávesy) generuje softwarové prerušenie. Prerušenie sa dá vyvolať aj inštrukciami int, int3 a into. Pokiaľ prerušenie nebolo vyvolané práve spomenutými inštrukciami, nazýva sa asynchrónne prerušenie. Pri príchode takéhoto prerušenia procesor preruší svoju činnosť a zavolá obsluhu daného prerušenia. Keď obsluha skončí, vykonávanie pokračuje buď opakovaným vykonaním inštrukcie, ktorá spôsobila výnimku, alebo nasledujúcou inštrukciou. Záleží od typu výnimky. Po prerušení od hardwaru (IRQ) alebo po softwarovom prerušení sa pokračuje nasledujúcou inštrukciou, kde bol procesor prerušený. Dokopy existuje 256 rôznych prerušení, z toho prvých 20 je využitých na obsluhu výnimiek procesora, ďalších 12 je na tieto účely rezervovaných. Zvyšné môžu byť využité v programe na volanie vlastných prerušení.

Pre každé prerušenie môže existovať obsluha tohto prerušenia. Pri vyvolaní prerušenia, procesor sa pozrie do tabuľky prerušení IDT a vyberie deskriptor pre dané číslo prerušenia. Riadenie je predané na tento deskriptor podobne ako pri volaní cez bránu.

Inicializácia spočíva vo vytvorení tabuľky s deskriptormi, ktoré ukazujú na obsluhy prerušení. Následne je lineárna adresa tejto tabuľky spolu s limitom uložená do registra IDTR pomocou privilegovanej inštrukcie LIDT.

Podľa typu výnimky sa správa aj obsluha prerušenia.

Hlásenie (Trap) - Procesor pri prerušení uloží návratovú adresu nasledujúcej inštrukcie. Obsluha výnimky môže niečo vykonať a po zavolaní IRET (ukončenie prerušenia) pokračuje vykonávanie prerušeného programu ďalej. Napríklad ladiaca výnimka, ktorá sa generuje po každej inštrukcií, ak je toto zapnuté príznakom v registri EFLAGS. Hlásenie značí, že sa udialo niečo, čo stojí za pozornosť, ale neohrozuje beh systému.

Chyba (Fault) - Procesor ukladá návratovú adresu práve vykonávanej inštrukcie, ktorá výnimku spôsobila a v niektorých prípadoch aj kód chyby. Obsluha výnimky musí chybu odstrániť, lebo po návrate z obsluhy sa pokračuje opäť previnilou inštrukciou a hrozí riziko zacyklenia. Riešením je aj prepísanie chybnej inštrukcie (to ale vyžaduje analýzu kódu, dosť fuška), alebo zmena návratovej adresy na zásobníku tak,

aby ukazovala na nejaké známe miesto v kóde a neskôr celý previnilý proces odstrániť z pamäte. Výnimka typu "chyba" znamená problém pri vykonávaní inštrukcie (napríklad chyba ochrany, neplatná inštrukcia, delenie nulou, ...). Chyba je však v každom prípade celkom dobre definovaná a obsluha má šancu odstrániť ju bez ohrozenia stavu celého systému. V programe Prot386 sú chyby zvyčajne ošetrené návratom na začiatok programu, čím prebehne celá inicializácia odznova.

Problém (Abort) - Chyba, ku ktorej došlo, je natoľko závažná, že nemá zmysel snažiť sa o jej odstránenie. Návratová adresa sa neukladá. Obslužná rutina má šancu zachrániť, čo sa dá a následne vykonať reset. Pokus o návrat na nejaké známe miesto v kóde je nezmysel, lebo stabilita systému je ohrozená. Typickým problémom je tzv. dvojitá výnimka, ktorá sa generuje v prípade, že počas obsluhy nejakej výnimky dôjde ku ďalšej vnorenej výnimke (nie všetky kombinácie výnimiek však spôsobujú dvojitú výnimku). Toto procesor nevydrží s nervami. Na zásobníku je uložený iba chybový kód 0, aby sme vedeli, že nás má procesor rád. IRET nemá cenu volať, tak tam tá nula ani veľmi neprekáža. Reset je to najvhodnejšie riešenie problémov.

6.3 Vytváranie deskriptorov

Program zobrazuje menu, v ktorom sa dá hýbať šípkami hore a dolu. Výber sa potvrdzuje klávesou ENTER a návrat o úroveň vyššie vždy zabezpečí klávesa ESC.

Práca s deskriptormi zahŕňa možnosti vytvoriť nový deskriptor alebo upraviť existujúci. Pri vytváraní sa nastavuje typ deskriptora, pri upravovaní sa nastavujú jeho atribúty. Pracuje sa nad ozajstnou tabuľkou GDT, preto po spustení vidno už nejaké vytvorené deskriptory, ktoré používa ku behu Prot386. Nachádza sa tu kódový a dátový segment aplikácie, tiež pomocné TSS deskriptory. Tieto sú pre beh a bezpečnosť programu potrebné, preto je ich zmena zakázaná. Taktiež nie je možné vytvárať deskriptory, ktoré obsahujú lineárne adresy menšie ako 1 MB. Tá časť pamäte je totiž vyhradená pre jadro a prípadné prepísanie obsahu by mohlo byť fatálne. Pre experimentovanie slúžia adresy od 1 MB vyššie.

6.4 Použitie deskriptorov

Keď sme sa dosýta vyhrali s deskriptormi, je na čase odskúšať systém ochrany. K tomuto účelu nám slúži menu "Execute code". Čo sa vlastne ide diať? V programe sú zahrnuté časti kódu, ktoré slúžia na testovanie a nazývajú sa

“moduly”. Užívateľ si vyberie modul, zvolí si parametre a nechá ho vykonať. Modul sa nakopíruje do vybraného segmentu a tento sa následne zavolá. Výsledky sa prejavia buď úspešným zbehnutím alebo chybou ochrany, o ktorú sa postará obsluha výnimiek a vypíše o tom správu. Pri každom module sa dá nastaviť počiatočné CPL, pod ktorým sa celé spustenie modulu vykoná. Prot386 beží pochopiteľne na CPL=0, čo sa nám nie vždy hodí - chceme experimentovať s rôznymi úrovňami. Nastaviť sa tiež dá RPL, s ktorým sa modul zavolá.

K dispozícii je jednoduchý modul, ktorý vykoná prerušenie s vybraným číslom, modul, ktorý číta alebo zapisuje z/do segmentu a do tretice modul, ktorý vykoná volanie a zobrazí aktuálne CPL. Práve tento posledný si rozoberieme bližsie.

Pri volaní (plnení CS registra) vstupujú do hry 3 hodnoty: aktuálne CPL, DPL volaného deskriptora a RPL selektora. Všetky 3 sa dajú nastaviť na 1 mieste a následne vykonať volanie. Opať nastaviť a opäť vykonať. Nastavenie modulu ukazuje podstatné detaily volaného segmentu a v prípade, že ide o volaciu bránu, zobrazia sa aj detaily segmentu, na ktorý brána odkazuje. V oboch sa dá meniť DPL, pretože u oboch záleží na jej hodnote. Tieto zmeny sa dajú robiť aj pri editácií deskriptora v prvom menu, ale tu sú všetky potrebné hodnoty pokope. Ak pri vykonaní nedôjde ku žiadnej výnimke, zobrazí sa CPL, na ktoré sa volanie dostalo (väčšinou to, ktoré si uživateľ na začiatku zvolil, ale niekedy aj nižšie - viď 5.2.3 a popis ako zmeniť svoje CPL). Ak hodnoty nevyhovujú systému ochrany, môžeme očakávať červenú tabuľku s chybovou hláškou.

6.5 Ako to presne funguje?

Volanie modulu nie je také jednoduché, ako sa na prvý pohľad môže zdať. Ako sa vôbec dá pred zavolaním kódu zmeniť CPL, zavolať a opäť sa vrátiť na CPL=0 nejako rozumne? Pri zmene CPL sa predsa mení aj zásobník. Ako to zvládnuť bez ohrozenia systému? Presne pre tieto účely existuje podpora multitaskingu priamo v procesore. Naše volanie modulu bude samostatná úloha (task). Takto budú v systéme 2 úlohy - Prot386 ako správca a volaný modul ako nejaký proces. Medzi úlohami sa dá jednoducho prepínať, každá úloha má svoj TSS deskriptor, kde je v čase nečinnosti uložený celý stav registrov tak, ako ho zanechala úloha tesne pred prepnutím. O toto sa opäť stará procesor.

Prot386 si na začiatku vytvára svoj TSS ako jedinú úlohu, ktorá v systéme beží. Pre účely volania modulov sa vytvorí nový TSS deskriptor, uložia sa doňho počiatočné hodnoty registrov (vrátane CS a hodnoty CPL) a pred volaním modulu sa prepne na tento nový TSS. Tu sa vykoná volanie segmentu

tak, ako si to užívateľ nastavil. Pokiaľ nastane výnimka, stačí sa vrátiť do rodičovského TSS. Po skončení volania sa takisto vráti riadenie pôvodnému TSS, čím sa obnoví stav sytému tak, ako bol pred volaním modulu. Pred každým volaním modulu je TSS deskriptor nanovo inicializovaný, čím sa zabudnú všetky problémy, ktoré pri predchádzajúcom volaní mohli nastať. Obe úlohy sú od seba úplne oddelené.

6.6 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).

Funkcia v C:

```
int suma(int a, int b) {
  int s;
  s = a + b;
  return s;
}
```

Volanie uvedenej funkcie v ASM:

```
push b
push a          ; ulozenie parametrov na zasobnik
call suma       ; volanie funkcie
addl $8, %esp   ; upratanie zasobnika (2*sizeof(int) = 8)
```

Prepis funkcie do ASM:

```
suma:
  push %ebp
  movl %esp, %ebp  ; vstupna sekvencia
  subl $4, %esp    ; vytvorenie priestoru pre 1 lokalnu premennu s
                   ; EBP obsahuje pointer na vrchol zasobnika
                   ; ESP ukazuje na lokalne premenne
```

```
  movl 8(%ebp), %eax   ; prvy parameter (a) sa ulozi do EAX
  movl 12(%ebp), %ecx  ; druhy parameter (b) do ECX
  addl %ecx, %eax      ; EAX:=EAX+ECX
                       ; navratova hodnota ostava v EAX
  movl %eax, -4(%ebp)  ; vysledok sa ulozi do premennej s

  movl %ebp, %esp ; vystupna sekvencia
  pop ebp         ; zrusi lokalne premenne zo zasobnika
  ret             ; vratenie riadenia
```

Prvé 3 riadky a posledné 3 riadky sa nazývajú vstupná a výstupná sekvencia. Je to zaužívaný štandard pre pristupovanie ku parametrom na zásobníku. Pre zjednodušenie kódu existuje inštrukcia ENTER, ktorá robí presne to, čo vstupná sekvencia a LEAVE, ktorá nahrádza výstupnú sekvenciu. Upravený program by teda mohol vyzserať takto:

```
suma:
  enter $0, $4

  movl 8(%ebp), %eax
  ...
  movl %eax, -4(%ebp)

  leave
  ret
```

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. CD médium je treba vypáliť s možnosou emulácie floppy diskety.

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. Pre výstup je určená funkcia write, ktorá vypíše reťazec na aktuálnu pozíciu kurzora. Poloha kurzora sa nastavuje cez zápis na I/O porty grafickej karty.

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.

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.

Efekt 8086 - viď Hradlo A20.

GDT/LDT - Globálna / Lokálna tabuľka deskriptorov[17] 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.

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.

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.

Inštrukcie 16/32 - Inštrukcie, ktorých význam je závislý od aktuálneho prostredia, lebo ich zápis je v oboch prípadoch rovnaký.

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 adresy, pripočítava sa ku bázovej adrese aby sme získali lineárnu adresu.

[17] angl. Global / Local Descriptor Table

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. Podľa typu: 1 kódový, 1 zásobníkový, 4 dátové.

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.

Tabuľka deskriptorov - viď GDT/LDT.

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.

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 Typy deskriptorov

A.1 Systémové deskriptory

Bit S = 0	
Typ - Popis	Typ - Popis
0000 - Reserved	1000 - Reserved
0001 - 286 Available TSS	1001 - 386 Available TSS
0010 - LDT	1010 - Reserved
0011 - 286 Busy TSS	1011 - 386 Busy TSS
0100 - 286 Call Gate	1100 - 386 Call Gate
0101 - Task Gate	1101 - Reserved
0110 - 286 Interrupt Gate	1110 - 386 Interrupt Gate
0111 - 286 Trap Gate	1111 - 386 Trap Gate

Deskriptory začínajúce 286 sú pre kompatibilitu s chráneným režimom procesorov 80286, kde bola štruktúra deskriptorov trocha iná ako na 80386.

A.2 Nesystémové deskriptory

Bit S = 1		
Bit	0	1
43	Data	Code
42	E	C
41	W	R
40	A	

E - Expand up - rastie hore (0), expand down - rastie nadol - vhodné pre zásobník (1).

W - Writable - len na čítanie (0), aj na zápis (1).

C - Conforming - neprispôsobený - volať možno len ak CPL = DPL (0), prispôsobený - volať možno aj ak CPL $\geq$ DPL (1).

R - Readable - len na vykonávanie (0), možno aj čítať (1).

A - Accessed - procesor nastaví na 1 pri každom prístupe.

B 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
make floppy
```

Posledný príkaz nahrá vytvorený image na floppy disketu, ak je nejaká vložená v mechanike.

Literatúra

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

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

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

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

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

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