KIV Operační systémy

IBM PC/AT & MS-DOS(-alike)

Verze 1.0

80286

- 1. ledna 1982
- 16-bitový procesor
  - Instruční sada x86-16
- První z rodiny x86 procesorů, který měl protected-mode používaný k izolaci jádra od uživatelských procesů

- Nicméně procesor po zapnutí nastartuje v tzv. real-mode, tj. stejně jako 8086 z roku 1978
  - Díky zachování zpětné kompatibility
    stejně jako dnešní nejmodernější x86-64

80186

- Vrstevník 80286, také vyráběný od roku 1982
  - Procesor pro vestavná zařízení
  - Instrukční sada x86-16
- Sice se už nevyrábí, ale zato se dodnes vyrábí RISCové procesory, které mají jeho instrukční sadu
  - HTL80186
  - VAutomation Turbo186
  - CAST C80186EC
  - Mentor Graphics M8086
  - iWave Systems 80186XL

80286 – vybrané registry

- AX, BX, CX a DX – obecné 16-bitové registry, které se dále dělí na dvojice 8-bitových registrů
  - AH, AL … DH, DL
- CS, DS, SS, ES – segmentové registry pro kód, data, zásobník a extra segmentový registr
- SI, DI, BP, SP – indexové registry; source, destination, base, stack
- Flags – stavový registr, výsledky operací
- MSW (386+: CR0-4, atd.) – stav CPU

80286 – adresace

- Adresa v paměti ji dána dvojicí registrů segment:index
  - Např. CS:IP ukazuje na instrukci, která se má vykonat
- Adresa má 20 bitů
  - Segment má 16 bitů
  - Index, tj. offset, má také 16 bitů
  - Adresa = (segment shl 4) or offset

	1100 1100 1100 1100	segment
+	0011 0011 0011 0011	offset
	1111 1111 1111 1111 0011	adresa

80286 – mapa paměti

Adresa	Popis
0000:0000	Tabulka vektorů přerušení; 256 4-bytových adres
0040:0000	Proměnné ROM-BIOSu
A000:0000	Paměť EGA; grafický režim obrazovky
B000:0000	Paměť MDA
B800:0000	Paměť CGA; textový režim obrazovky
C800:0000 až E000:0000	Externí ROM, ROM-BIOS v ní hledá spustitelný kód – ovladače zařízení
E000:0000 až FE00:000	ROM-BIOS: POST, defaultní obsluhy přerušení, SETUP, diagnostika, atd.
F000:FFF0	Instrukce JMP, která se skočí na první instrukci, která se provede po zapnutí/resetu procesoru
F000:FFF5	Datum verze BIOSu
F000:FFFE	Identifikační kód PC

MS-DOS

- Disk Operating System
- Jeden z nejvýznamnějších (tj. ne nutně nejlepších) operačních systémů
- Poskytuje konzoli a souborový systém
  - Další je záležitost ovladačů a náhrady/nástavby „shellu“ – např. Windows 3.1
- Existovalo mnoho DOSů
  - MS-DOS, PC-DOS, DR-DOS, QDOS…
  - MSDOS.SYS, IO.SYS a COMMAND.COM se pak jmenují jinak
  - A dodnes aktivní FreeDOS

80286 – MS-DOS Bootstrap

1. Po zapnutí/hw resetu (tj. hot reset počítače) se procesor uvede do aktivního stavu a do režimu real-mode, tzn. má přístup pouze k 1MB paměti a kód může číst a zapisovat na libovolné místo v paměti
2. Registry CS:IP se nastaví na hodnoty F000:FFF0 a CPU začne vykonávat instrukce od této adresy
   - Tj. skočí na první instrukci ROM-BIOSu, čímž spustí jeho kód – proto se této adrese říká reset vector
   - Cold reset počítače: předání se řízení na adresu určenou reset-vectorem; od 386 je to fyz. lineární adr. 0xFFFFFFF0

80286 – MS-DOS Bootstrap

3. Podle nakonfigurovaného pořadí BIOS hledá disky
4. Při nalezení prvního disku BIOS načte do paměti, adresa 0x7c00, jeho první sektor, tj. prvních 512 bytů, předá řízení CPU na tuto adresu – tj. nastaví CS:IP
   - První sektor se nazývá Master Boot Record (MBR)

Velikost	Popis
440B	Kód
4 B	Signatura disku
2B	0x00 0x00 -
4x16 B	Tabulka rozdělení disku
2B	Signatura MBR

80286 – MS-DOS Bootstrap

5. Kód načtený z MBR má za úkol načíst zbytek zavaděče ze správného/aktivního/vybraného oddílu disku
   - Může být zavaděč přímo operačního systému
   - Anebo manažer, který dá vybrat, který OS se má zavést, je-li jich nainstalováno více
   - Anebo to také může být vir, který infikuje počítač ještě před načtením OS
   - MS-DOS startuje z FAT, active&bootable&primary oddílu

80286 – MS-DOS Bootstrap

6. Poté, co kód z MBR identifikoval použitelný diskový oddíl, pokračuje s načítáním OS do paměti
   - Jedná se o soubory IO.SYS a MSDOS.SYS, které musí být uloženy kontinuálně na začátku oddílu
7. Jakmile jsou sobory načteny, řízení převezme IO.SYS
   - Rozhraní mezi DOSem a I/O subsystémem, které zpřístupní základní periferie
8. IO.SYS předá řízení MSDOS.SYS
   - Jádro OS, které poskytuje abstrakci od HW pomocí poskytovaných služeb

80286 – MS-DOS Bootstrap

9. Pokud existuje, MSDOS.SYS načte a zparsuje CONFIG.SYS
   - Zavede ovladače paměti (XMS, EMS), periferií (CDROM, Myš, zvuková karta, atd.)…
10. MS-DOS.SYS načte a spustí, tj. předá řízení, COMMAND.COM (interpret příkazů, aneb shell)
11. Pokud existuje, spustí se AUTOEXEC.BAT
    - Dávkový soubor s příkazy pro COMMAND.COM
12. C:\ aneb Hotovo!

Windows -9X Bootstrap

- 16 bitové Windows se spouštěly příkazem win.com
  - Zavaděč Windows/386 přepnul procesor do tzv. protected-mode
- Windows 9x se zaváděly tak, že IO.SYS provedl konfiguraci počítače v real-mode a pak spustil win.com po dokončení AUTOEXEC.BAT
  - Komplikované zavedení ovladačů, některá zařízení mají ovladače jen pro real-mode, zatímco Windows běží v protected-mode => potřeba virtualizace – V86 mode

Jádro větší než 1MB

- Např. Linuxové jádro může být větší než 1MB
- Jenomže CPU nastartuje v real-mode s 20-bitovou adresou
- Jak zavést tak velké jádro?
  - 386+ procesory lze přepnout do tzv. unreal-mode, který zpřístupní dostupnou paměť jako 4GB flat address space
    - A pak lze načíst a spustit jádro větší než 1MB
  - Anebo máme UEFI – což nikdy nebyl případ MS-DOSu
    - Ale stále existuje např. FreeDOS

Unreal mode

- Není to oficiálně garantovaná vlastnost, ale existuje protože ji potřebuje System Management Mode
- Aby mohly programy běžet v protected-mode (32-bitová adresa, segmentace, stránkování, izolace procesů), je třeba vytvořit GDT a LDT
  - Globální a lokální tabulky deskriptorů segmentů – organizace paměti
- Segmentům se nastaví maximální velikost (limity) a procesor se přepne zpět do real-mode
  - Respektive do unreal-mode protože limity zůstanou zachovány

Příliš velký program (DOS)

- Co když máme program pro real-mode, který ale vyžaduje více paměti, než kolik jí je volné?
  - XMS, EMS – paměťové manažery, které přepnuly procesor do protected mode a na rozdíl od unreal mode v něm zůstaly
  - Real-mode programům, pak umožnily využít větší paměť po max 64kB velkých oknech, které kopírovaly mezi adresami nad a pod 1MB
- Programy pak dynamicky překrývaly blok paměti různými částmi, moduly, svého kódu => tzv. overlays

Overlays

- Technika, která umožňuje spustit program, který je větší než velikost dostupné paměti
  - Předchůdce virtuální paměti (příští přednáška)
  - Stále se používá u vestavěných zařízení, kde virtuální paměť není k dispozici
- Program se rozdělí do funkčních modulů, overlays, které mají stromovou strukturu
  - V paměti mohou být zavedeny jenom moduly od na cestě kořene, tj. fce main, až k listům
  - Návrh stromů a explicitní zavádění a uvolňování musí zařídit programátor

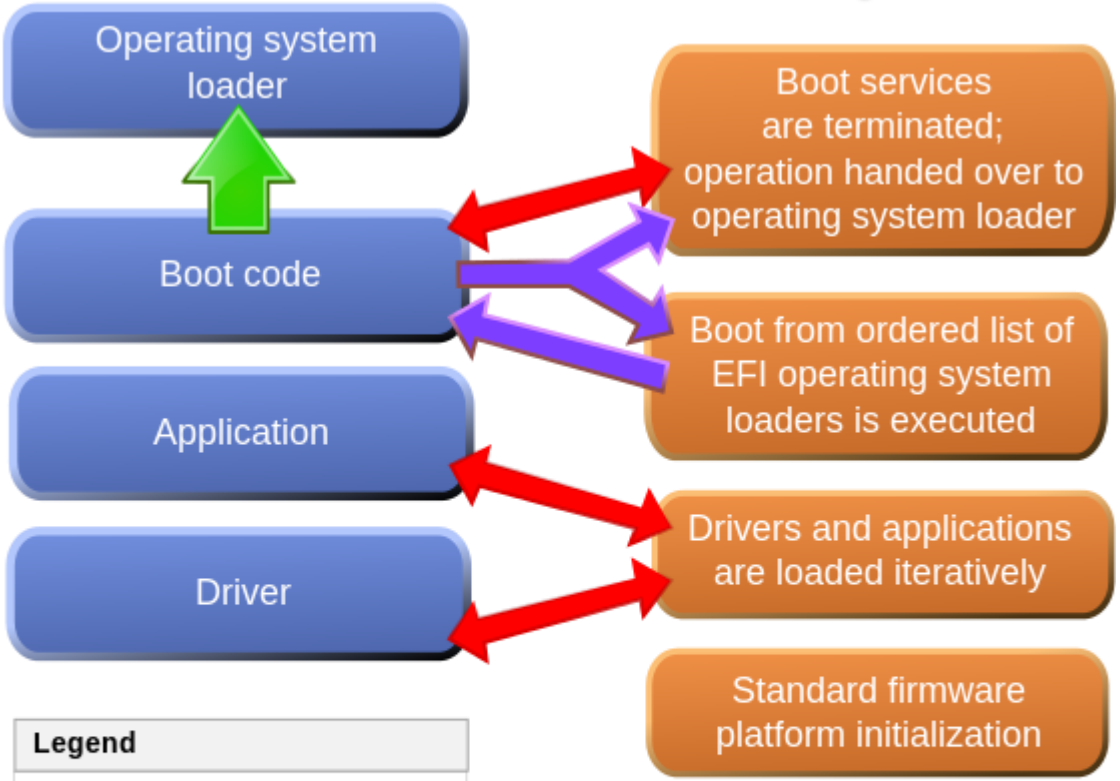
UEFI

- Unified Extensible Firmware Interface
- Nástupce BIOSu, který má řešit jeho nedostatky
- Nespoléhá se na boot (tj. první) sektor, ale definuje boot manager
  - Umí zavádět pouze důvěryhodně podepsaný kód – což nejsou např. viry – BIOS uměl zamezit přepsání MBR
  - Nicméně má legacy mode, ve kterém se chová jako BIOS
- Umí přepnout procesor do cílového režimu, např. protected-mode – zavaděč OS to ale musí očekávat

UEFI - pojmy

- EFI executable – spustitelný soubor v UEFI bytecodu (ne Java bytecode)!
- Guid Partition Table (GPT) – načisto udělaná tabulka diskových oddílů, tak aby se nemusel vláčet omezující balast z minulosti
- EFI System Partition – oddíl naformátovaný souborovým systémem FAT dle specifikace UEFI
- Umíme načíst a spustit soubor z disku – tj. stejný princip jako doposud, jen jiná, novější specifikace

UEFI - Bootstrap



- UEFI načítá ovladače stejně jako to dělal ROM-BIOS a IO.SYS

http://en.wikipedia.org/wiki/File:Efi_flowchart_extended.svg

MS-DOS API

- Chce-li program zavolat službu operačního systému, de-facto volá požadovanou rutinu, která je už někde v paměti – ale jak ji najde?
- Řešením je, aby byla na předem známé adrese, kam se předá řízení procesoru nastavením CS:IP
- Rutin implementujících jednotlivé služby může být mnoho => použije se další registr, např. ax na určení konkrétní služby

MS-DOS volání API

- Služba MS-DOS se zavolá pomocí přerušení 21h
  - Tzn. adresa hlavní rutiny služeb OS je zapsána na 0x21. pozici tabulky vektorů přerušení
  - Např. zjištění verze DOSu vypadá následovně

```
mov ah, 0x30h
int 21h
;po návratu jsou hlavní a vedlejší čísla verze v AL a AH
```

MS-DOS obsluha API

1. Program nastaví příslušné registry vykoná int 21h
2. Do zásobníku, na jehož vrchol ukazuje SS:SP, se uloží registry CS:IP (ukazující ve volajícím programu na další instrukci po int 21h) a registr Flags
   - Provede procesor v rámci zpracování instrukce int 21h
3. Jádro OS získá kontrolu nad CPU a vidí všechny registry volajícího programu
4. Jádro OS provede příslušnou akci a nastaví příslušní registry podle výsledku akce

EGA-BIOS

- Co když budeme chtít využít služeb, které neposkytuje BIOS, ale např. grafický adaptér?
- Např. budeme chtít změnit režim obrazovky na 320x200x256. Program vykoná následující instrukce

```
mov ah, 0       ;služba Změň videorežim
mov al, 13h     ;režim 320x200 256 barev
int 10h
```

- Z hlediska procesoru se stane to samé jako při instrukci int 21h, pouze vykonávaný kód bude v paměti někde jinde

Zápis do video paměti

- Na adrese a000:0000 je první pixel právě nastaveného videorežimu, levý horní roh
- Na každý pixel je jeden byte, byte je index do palety barev
- Přímým zápisem do namapované videopaměti měníme barvy jednotlivých pixelů
- V textovém režimu je znak v levém horním rohu na b800:0000
  - Má dva byty – 1. byte kód znaku, 2. byte atributy, např. barva

VESA

- Videorežim 320x200x256 potřebuje méně než 64kB na uložení indexů pixelů do palety barev
- Videorežim 640x480x16 také
- Ale videorežim 640x480x256 už ne – navíc to dříve nebýval standardní režim, dokud se neobjevila specifikace VESA - SuperVGA
- Před VESA sice bylo možné na některých kartách takový režim aktivovat, ale postup byl proprietární
- Proto vznikl rezidentní software, který VESA emuloval

VESA - stránkování

- Video režim 1280x1024 při 24-bitové barevné hloubce potřebuje více než 64kB paměti
- Řešením bylo stránkování
  - Obraz se rozdělil na několik stránek po 64kB
  - 64kB od A000:0000 umožňovalo přímý zápis a čtení do aktivní stránky
- Aktivní stránka se zvolí funkcí VESA BIOSu
  - Buď pomocí int 10h – opakované volání int 10h je pomalé
  - Anebo call na konkrétní adresu obslužné rutiny; adresu získáme přes int 10h – call je daleko rychlejší než int, který je potřeba pouze jednou

VESA – Linear Frame Buffer

- I když je stránkování video paměti pomocí call rychlejší než pomocí int, je stále pomalé
- Rychlejší je přímý přístup do video paměti, jako tomu bylo u videorežimů, kterým stačilo 64kB
- VESA umožňuje získat adresu Linear Frame Buffer (LFB)
  - Obdoba A000:0000 u EGA, ale adresa LFB není pevně daná
  - Je třeba LFB „povolit“ a následně získat adresu od grafické karty
  - Blok paměti, který nelze použít pro data a kód programů

Zřetězení obsluhy přerušení

- Např. chceme-li sw emulovat VESA

1. Program si do vlastní proměnné načte adresu stávajícího vektoru přerušení – int 10h u sw VESA
2. Program zapíše do tabulky vektorů přerušení adresu své rutiny, která bude přerušení nově obsluhovat
   - Např. u sw VESA obsluhuje služby s AH=4Fh (VESA extension)
   - Nová rutina přerušení může, případně musí (např. int 08h hodiny), zavolat (už ne int!) předcházející obsluhu přerušení
3. Program skončí službou OS Terminate and Stay Resident (TSR)

Ukončení TSR

- TSR program měl smysl pouze tehdy, pokud obsluhoval některé přerušení
- V paměti mohl být načteno několik TSR obsluhujících stejné přerušení
- Ale co když se měl jeden z nich ukončit, a nebyl to ten poslední?
  - Byl to problém, protože neexistoval standardizovaný protokol, jak vyjmout ze zřetězeného seznamu obsluh přerušení někoho uprostřed

Interrupt request (IRQ)

- Instrukce int je sw-vyvolané přerušení
- Pokud přerušení vyvolá hw, pak se bavíme o IRQ
  - Každé IRQ má svoji prioritu – Level aka IRQL
- Při IRQ procesor zastaví vykonávání aktuálního programu, uloží Flags, CS a IP, a začne vykonávat příslušenou obsluhu přerušení dle tabulky vektorů přerušení
- Programmable Interrupt Controller (PIC) mj. překládá číslo IRQ na index do tabulky vektorů přerušení

Časovač

- Např. tik hodin je IRQ0 (nelze změnit ani maskovat) a vyvolá obsluhu přerušení int 08h
  - Proběhne každých 55ms
- Na tomto přerušení závisí mnoho důležitých činností a je proto nutné, aby
  a) Bylo co nejrychlejší
  b) Zavolalo původní obsluhu přerušení
- Pomocí časovače se implementuje preemptivní multithreading (a následně multitasking)

Časovač - implementace

- Nejprve se do proměnné oldVec8 uloží adresa původní obsluhy přerušení, takže obsluha může vypadat následovně:

```
pushf                              ;simulace volání obsluhy přerušení
call dword ptr cs:[oldVec8] ;pro původní obsluhu
…                                  ;naše vlastní činnost
iret                               ;návrat do přerušeného programu
```

I/O Porty

- Input/Ouput base address – adresa prvního portu
- Periferie lze také ovládat pomocí portů – jedno zda blikáme s LED klávesnice, nebo programujeme PIC
  - Zápis odešle příkaz; instrukce out
  - Čtení čte stav nebo výsledek operace; instrukce in
- Např. při obsluze časově závislých činností v int 08h je nutné poslat řadiči přerušení informaci, že přerušení již skončilo

```
mov al, 20h          ;signál Konec přerušení
out 20h, al          ;port řadiče přerušení 8259
```

Viry

- MS-DOS umožnil ovládat počítač a jeho periferie
- Ale špatně nebo záměrně napsaný program mohl číst a přepisovat jeho vnitřní proměnné, a libovolně měnit činnost systému
  - Např. na přerušení se mohl pověsit vir, který se spouštěl z infikovaného MBR disku a infikoval MBR disket, a díky zavedení před jádrem OS mohl utajit své soubory na disku filtrováním systémových volání
    - Antiviry musely skenovat paměť, což byla příležitost pro polymorfní viry, které měnily svůj kód v paměti za běhu