Operační systémy

16. prosince 2008
10:43

První vrstva ležící nad HW - OS

Počítače 1. generace
- Programování ve strojovém kódu; propojováním low level částí OS drátama...
- Technologie: relé, elektronky
- Propojovací desky, děrné štítky
- Operační systém
  - Koncept OS neexistuje
  - Konstruktér, programátor, správce v jedné osobě
- Od začátku už specificky stavěn na nějaký úkol

Počítače 2. generace
- Technologie: tranzistory, pevný disk (50 disků, 61cm průměr, 5MB)
- Programování: strojový kód, už programovací jazyk Fortran
- Operační systém
  - Začíná se objevovat koncept SW, který už je z výroby, a umí už spouštět programy, které vesměs přišli na děrném štítku; dá se to přirovnat k BIOSu, umí ovládat vstupy, dá se spouštět,
  - Sekvenční spouštění programů, omezení doby běhu - vynuceno HW, aby se ten strojový čas dal trošku sdílet, úlohy se naskládaly za sebe
  - Běžící program má plnou kontrolu nad počítačem
  - Objevuje se první time-sharing systém
    - Docílíme toho, že se sdílí výpočetní čas a s kratšími úlohami se nemusí čekat, až doběhnou delší
      - Compatible Time Sharing Systém (IBM 7094 + HW mod - časovač navíc, který určoval přepínání procesů)

Počítače 3. generace (60. - 70. léta)
- Technologie: integrované obvody nízké integrace; myš; DRAM; ARPANET
- Programování: BASIC, Pascal, SmallTalk, C
- OS
  - TSS, OS/360 Multics, unics/unix
  - Multitasking, spooling, virtual memory & machine
  - Memory mapped files [po paměti možno chodit pointereama a to se zpět promítá do souboru], dynamic linking [sdílené knihovny]
  - Opraváři, systémoví a aplikační programátoři

Počítače 4. generace (od 80. let)
- Tech: integrované obvody vysoké generace > vznik osobních počítačů (IBM PC, ZS Spectrum…)
- Programování: opět BASIC, Pascal, SmaillTalk, C
- Operační systém
  - Cokoliv mezi zavaděčem programu a dustrubuovaným, více uživatelským a více programovým OS (Finder, CP/M, DOS..) - pro osobní počítače
  - Unix, Windows, MacOS, Linux .. Už berou i prvky z mainframe počítačů - vznik s potřebou správou paměti, procesorového času

Hlavní funkce OS

Extended machine
- Poskytovat rozšířený stroj, který programátorovi umožní nezabývat se drobnými detaily při konstrukci počítače; aplikace jsou to, co dělá business - snaha o rychlý vývoj aplikací
- Oddělujeme aplikace od HW platformy
- Poskytuje abstrakce oddělující HW, na HW nezávislé - soubory, sockety pro síť
- Program už nemusí komunikovat přímo HW, to mu poskytne OS

Resource manager
- Správce přostředků, který zajišťuje vzájemnou izolaci aplikací, které běží na tom stroji
- Aby si aplikace nemohly vzájemně škodit, ani v tom nemusíme hledat úmysl, spíš jako ochrana proti chybám programátora
- Řeší přidělování a sdílení prostředků
  - Prostředky: CPU - čas procesoru, RAM - velikost, přenosové pásmo - traffic, disková kapacita

Architektura
- OS je obyčejný program jako každý jiný - píše se stejně jako jiné, jen je větší nárok na správnost
- Liší se v tom, že OS je ten program, kterému věříme - běží v privilegované roli, ostatní aplikace už běží ve výrazně méně privilegované roli
- Musí poskytovat abstrakce - soubory, sockety
  - Mnoho subsystémů pro různé fce - navrženo modulárně, aby se dali programovat současně, nezávisle udržovat, dala se zvládnout se složitost
    - Rozdělení zodpovědnosti, zjednodušení návrhu
    - Různá oprávnění pro různé subsystémy - zvýšená odolnost proti "zlým" programům
      - Ne všechny subsystémy potřebují komunikovat navzájem, proto si můžeme dovolit tyto různé úrovně oprávnění
  - Ta možnost dát subsystémům různá oprávnění, aby si navzájem neškodili (může být díra v systému) vede k dilematu při návrhu

Dilema při volbě architektury
- Oddělení subsystémů vs. Efektivita OS
  - Umělé bariéry mezi subsystémy vyžadují další režii pro kontrolu správnosti komunikace
  - Pro ochranu, aby tento systém nešel obejít, tak musíme využít i HW podporu procesoru - souvisí s oprávněními, co se dá na procesoru - pro zaslání zprávy jiného subsystému, musíme požádat část systému, která má vyšší oprávnění, aby to zprostředkovala - musíme přepnout procesor do stavu, kdy dojde k přepnutí úrovně oprávnění - tohle přepínání je poměrně drahá záležitost, a komunikace není zrovna intuitivní
  - Výsledkem je ale velice robustní architektura
- Kompromis mezi odolností a efektivitou
  - Rozdělíme jen na různé funkce, na moduly, balíčky
  - To znamená, že komunikace není nikým kontrolována a není vynucována, závisí to na tom, jestli si vývojáři jádra důvěřují, a nic nebrání nikomu komunikovat s jakýmkoliv subsystémem.
  - Pak je třeba si hlídat, když do jádra natahujeme kód třetích stran (u Windows údajně zdroj nejvíce bluescreenů) - ovladače
  - Očekává se, že kód v OS se bude chovat slušně, monolitičnost kódu spočívá v tom, všechny části systému mají stejná oprávnění a jsou si rovna = nízká režie na komunikaci, netřeba pokaždé přepínat kontexty procesoru

  Dnešní systémy dávají vesměs přednost efektivitě před robustností - monolitické systémy

  Uživatelské programy běží v dost omezeném rozsahu oprávnění; mají možnost volat části systému pomocí speciálních instrukcí, procesor se přepne na nižší úroveň, zkontrolujeme, jestli to co aplikační program chce je vůbec možné

  Toto se nepoužívalo v zavaděčích typu msdos - tam je vše sdílené a jeden program může celý systém shodit
  - Pro msdos je limit paměti 640kB - nad tím už je mapována videopamět, prázdné místo a na konci je namapován, načten z ROMky, bios
  - MS dos používal pro přístup k disku přímo podporu BIOSu; uměl načíst aplikaci do paměti, a pak ji spustit; v MS DOSu mohla paměť psát kamkoliv
  - Bios najde disk, který zvolíme.. Na něm najde bootsector, který má za úkol najít OS, nyní se v bootsectoru dá bootloader, ten už umí procházet i disky, načte soubory do paměti a spustí je, pak už běží OS - ten opět musí detekovat, co má vlastně za HW

Systémové volání
- Způsob, kdy žádáme vyšší autoritu a provedení něčeho

- Například otevření souboru:
  - v jazyce máme např. příkaz OPEN, ten zajistí, že přejdeme z úrovně uživatelských aplikací do chráněné oblasti systému, systém zjistí, co chceme, podívá se, jestli to vůbec jde (jestli soubor existuje, jestli jsou práva…) a systém vrátí zpět identifikátor pro případnou další operaci se souborem.. Kdyby ale program chtěl zkoušet něco víc, tak bude sestřelen - neoprávněný přístup

  Příkaz: Open (….)
  Instrukce pro přečtení [jen příklad]:
  MOV EAX, 01
  MOV EDX, &filename
  INT 80h ..

  Instrukční sada intelu

  INT .. interrupt

  Zde se používá stejné číslo, dáno konvencí
- Jediný způsob, jak program může požádat o nějakou systémovou službu

Aplikace
Systémová volání
Subsystémy jádra
Ovladače HW
HW

Tučně.. Kernel API

Systémy dávající přednost robustnosti - založeny na mikrojádře
- Subsystémy mají minimální oprávnění, vlastně podobné aplikačním programům
- Výsledkem je minimální jádro, plus nějaká sada systémových serverů zajišťující služby subsystémů
- Jádro poskytuje jen základní fce: zprostředkuje přerušení, komunikaci mezi programy, zajišťuje kontrolu oprávnění
- Jádro, kód kterému musíme věřit, je strašně malé, v řádu tisíců řádků (vs. Linux, který má miliony kódů)
- Výzkumné a embedded systémy [QNX, L4 - u něj není ta režie tak veliká, stačí ji správně napsat]
  - MACH, Spring, minix
- Důraz na spolehlivost: Trusted Computing Base - TCB! :-) [viz. OI I.]
- Volání služeb: transparentní komunikace v síťovém protředí
  - Zde je snadný rozdíl, jestli ta komunikace je v rámci jednoho stroje, nebo po síti, už to tak je navrženo.. Logicky to je identické, jen to může mít větší latenci
- Windows NT 3.51 se ještě docela podobaly Mikrokernelu.. Cele win32 api bylo v user modu místo v kernelu
  - Měli nad jádrem OS i dva subsystémy - Win API a Posix API, což umožňovalo i docela širokou podporu pro unixové aplikace
    - Snaha o systém, který by podporoval širokou bázi programů

Virtualizace HW
- Opět nám tu vzniká mezivrstva, na jednom fyzickém stroji můžeme pustit více OS zároveň - snaha o maximální využití stroje
- Původní podoba už na IBM/360, abstrakce HW + plánování běhu virtuálních strojů
- Dnešní podoba:
  - Abstraktní stroj bez závislosti na HW
    - AS 400 [IBM], Java, CLR (.NET)
  - Virtualizace na holém HW i v hostitelském OS - hypervizor - tenká vrstva, která virtualizuje nejzákladnější služby (přerušení, časovač, správu paměti) a OS si tak myslí, že vlastní cely HW
    - Vmaware, xen,...