IB001 Úvod do informačních technologií – podzim 2013

1. přednáška

Společenské aspekty:

- výrobní a obchodní procesy
- nástroj vědy
- komunikace
- zábava
- kriminální činnost

Výrobní a obchodní procesy:

- Řízení výrobních procesů – vcelku stará disciplína, existují už několik desítek let. Systémy pracující v reálném čase, je třeba zohlednit právě čas.
- Informační a manažerské systémy (IS)
- Nové formy vývoje (matematické modelování výrobků před jejich samotnou výrobou; simulace místo fyzických modelů; aerodynamické tunely – nejprve propočet, teprve potom měření)
- Ovlivnění forem spolupráce/komunikace (např. reklamy)
    - Mezi institucemi (B2B, Bussiness to Bussiness)
    - Instituce a zákazník (B2C, Bussiness to Customer)
    - Mezi zákazníky (C2C)
- Zcela nové příležitosti (Google, mapy, GPS, …) – bez počítačů a internetu by nic z toho nebylo možné/tak masové

Nástroj vědy:

- Původní použití počítačů – vzájemné partnerství, věda tlačila vývoj počítačů dopředu, ale i naopak
- Trvale klíčový směr využití
- Ovlivňuje způsob vědecké práce – citace daného autora: čím více je citován v jiných vědeckých pracích, tím se zvyšuje jeho prestiž; citování (a sledování citací na sebe) je mnohem snazší.
    - Experimenty versus simulace – experimenty v dnešní době většinou už jen potvrzují to, co nejprve udělaly simulace
    - Statistické zpracování velkých souborů - třeba CERN, hledání Higgsonova bosonu (tzv. božské částice)
        - Astronomie
        - Bio-informatika
    - Virtuální vědecké týmy (spolupráce) – propojení vědců na druhém konci světa. Pouze virtuální kontakt někdy nestačí (videokonference).
- Formule 1 výpočetní techniky

Komunikace:

- Komunikace mezi počítači – řízení počítačů na velkou vzdálenost
- Komunikace mezi lidmi (případně člověk-automat) – opět roste význam
    - Telefony
    - Faxy
    - Mobilní komunikace
- Média – např. dříve pouze „kus papíru“, dnes už jsou běžné čtečky e-knih
- Zvýšení fragility společnosti – ne všichni tento pokrok akceptují, nebo stíhají; mezigenerační konflikty a nedorozumění, skupiny obyvatel vyloučené ze společnosti

Zábava:

- Televize – dnes již upadá, není tolik interaktivní, chybí jim mladé publikum
- Počítačové hry
    - Fenomén on-line her: specifické prostředí pro spolupráci
- Pasivní versus aktivní přístup – velký výběr
- Peer to peer sítě (Napster, Gnutella, …) – např. stahování hudby, filmu – konflikt s ochránci autorských práv, nemožnost kontroly
- Virtuální realita

Kriminální činnost:

- Tzv. „*kriminalita bílých límečků*“
- Zneužívá zdrojů na síti (účty, výpočetní výkon, kapacita sítě, poštovní služby, …)
- Krádeže informací (čísla kreditních karet, telefonní linky, špionážní činnost) – dokonce existují i burzy, na kterých se dají tato ukradená data nelegálně koupit
- Viry
- Záměrně špatné informace – velice snadná implementace
- Destabilizace společnosti – např. i těmi záměrně špatnými situacemi, aféra na ČT při nabourání přímého přenosu a zobrazení jaderného výbuchu v Krkonoších
    - Specifickým šířením informací
    - Útoky na infrastrukturu
    - Útoky na citlivé informační zdroje

Právo a etika v IT

- Mladá věda, tudíž zde právo a etika nejsou příliš ustálené. Stále se vyvíjí – bude se pravděpodobně měnit nejen za našeho studia, ale i dále
- V podstatě inženýrská disciplína, avšak neinženýrské přístupy (shrink wrap license, minimální odpovědnost za chyby, …)
    - shrink wrap license = licence přijatá otevřením obalu. Druh licence, jejíž podmínky kupující přijímá tím, že roztrhne krabici nebo obal s instalačními disky.
- Kódy/normy správného chování/přístupu
- Faktická a právní odpovědnost
- IPR (Intellectual Property Rights), autorská ochrana, softwarové patenty

Dekompozice

- Dva rozměry – nejlepší je si problém rozdělit na podúkoly, „vrstvy“:
    - Od fyzické po programovou vrstvu
    - Různé komponenty na téže vrstvě
- Příklady:
    - ISO/OSI síťový model: např. linková, transportní a aplikační vrstva
    - Vrstvy operačního systému: např. kernel, ovladače, překladače, aplikace
    - Různé typy procesorů
    - Různé programovací jazyky

Disciplíny

- Technické prostředky
    - Architektura počítačů a sítí
- Programové prostředky
    - Operační systémy
    - Programovací jazyky
    - Aplikace

Technické prostředky – základní pojmy

- Procesor (- řadič) – paměť – periferie: **von Neumannova architektura**
    - schéma počítače, které používá jednu sběrnici, na kterou jsou připojeny všechny aktivní prvky (procesor, paměť, vstupy a výstupy).

      Popisuje počítač se společnou pamětí pro instrukce i data. To znamená, že zpracování je sekvenční oproti například harvardské architektuře, která je typickým představitelem paralelního zpracování.

      Procesor počítače se skládá z řídící a výkonné (aritmeticko-logické) jednotky. Řídící jednotka zpracovává jednotlivé instrukce uložené v paměti, přičemž jejich vlastní provádění nad daty má na starosti aritmeticko-logická jednotka. Vstup a výstup dat zajišťují vstupní a výstupní jednotky.
- Řízené zpracování dat
- Jiné modely architektury:
    - Turingovy stroje
    - Dataflow přístup (datové toky)
    - Objektově-orientovaná
    - Deklarativní (funkcionální či logická)

Procesor

- Stroj vykonávající instrukce
- Vnitřní hodiny, takt procesoru
- Základní jednotka sekvenční (**ALU** – arithmetical logical unit)
- Může obsahovat více jednotek: vnitřní paralelismus
- Instrukční cyklus: výběr a provedení instrukce jednou jednotkou

Typy procesorů

- Univerzální
  - **CISC**: Complex Instruction Set Computer
  - **RISC**: Reduces Instruction Set Computer
    - CISC: Označení complex vyjadřuje skutečnost, že strojové instrukce pokrývají velmi široký okruh funkcí, které by jinak šly naprogramovat pomocí jednodušších již obsažených strojových instrukcí (například násobení je možné nahradit sčítáním a bitovými posuny). Opakem procesorů CISC jsou procesory RISC, které obsahují redukovanou instrukční sadu.
- Specializované
  - Vektorové
  - Grafické
  - Embedded
  - …

Paměť (vnitřní)

- Uchovává data
- Přímo adresovatelná: sloupec a řádek (jako matice)
  - Rozsah adres: 16, 32, 64, … bitů
- Cyklus paměti: doba nezbytná pro vystavení nebo zápis dat
- Vzpamatovávání se po provedené operaci, prokládání pamětí
- *Statická* vs. *dynamická paměť*, *volatilita* = ztrácí se data při odpojení od sítě el. energie
- Hierarchie pamětí – čím je program důležitější, tím vyšší prioritu má
  - Rychlá-pomalá
  - Drahá-levná

Periferie

- Zajišťují vstup/výstup informací:
  - Komunikace s uživatelem
  - Permanentní ukládání dat – co je to permanentní? → informace tesané do kamene. Nynější nosiče mohou být za pár let passé – např. dnes už se příliš nepoužívají magnetické pásky a nejsou zařízení, na kterých by šly přečíst, případně data dále předat počítači.
  - Komunikace s jinými systémy

Komunikace s uživatelem

- Interaktivní
  - Klávesnice: vstup
  - Myš, tablet, stylus, …: vstup
  - Obrazovka: vstup i výstup (tablet)
  - Zvuk: vstup i výstup
- Dávkové: prostřednictvím jiných zařízení

Permanentní ukládání dat

- Paměti (ROM – read only memory, PROM – programmable ROM, EPROM – erasable PROM, NVRAM – non-volatile RAM (uchová informace i po odpojení napájení))
- Disky
    - Magnetické
    - Magnetooptické
    - Optické
- Pásky – jsou odolné (černé skříňky letadel; odolné vůči agresivnímu prostředí, ohni, …), nepotřebují elektriku; používají je spíše než klasičtí uživatelé pouze velká centra
- Sítě
- Papír – dnešní novinový papír nevydrží tolik, co dříve

Komunikace

- Počítačové sítě
    - Drátové
        - Elektrické
        - Optické
    - Bezdrátové
        - Radiové vlny
        - Optické
        - Akustické
- Mechanické

Speciální periferie

- Virtuální realita
    - Brýle a helmy
    - 3D projekce a prostorový zvuk
    - Haptika (rukavice, …)
    - Detekce polohy a pohybu
- Wearable computers

Co je to počítač?

- Standartní pohled:
    - Procesor(y)
    - Paměť
    - Periferie
- Možné i jiné pohledy
    - Buněčné automaty
    - Neuronové počítače
    - …

Paralelní systémy

- Úzce propojené (tightly coupled)
- Volně propojené (loosely coupled)
- Distribuované
- Gridy

Úzce propojené systémy

- Často **společná paměť**
- Minimální vliv vzdálenosti procesorů
- Speciální propojení procesorů a pamětí
- Vhodné pro tzv. *jemný paralelismus* – proces se rozdělí na úkoly, které se vloží do sdílené paměti, může k nim přistupovat více procesorů naráz
- Typický výpočetní model: **sdílená paměť** (i kdyby byla pouze virtuální)

Volně propojené systémy

- Převážně **distribuovaná paměť** (každý procesor zvlášť) – procesor A umí sáhnout do paměti procesoru B (trvá déle, než sáhnout do své paměti) – nebo ani to neumí a musí si zažádat o „povolení“ od procesoru B
- Vzdálenost procesorů může hrát roli
- Speciální propojení procesorů
- Výrazně vyšší latence (zpoždění) v meziprocesorové komunikaci (jednotky mus a méně) – s vysokou latencí se musí počítat už při programování
- Existence operací remote put a remote get pro přístup do paměti vzdáleného procesoru
- Typický výpočetní model: **zasílání zpráv**

Distribuované systémy

- Rozšíření předchozího modelu
- Vždy distribuovaná paměť
- Vzdálenost procesorů hraje významnou roli – ale i třeba v měřítku států, Evropy, …
- Propojení procesorů často formou běžné LAN sítě – propojují se v podstatě celé počítače – clustery (servery)
- Vysoká latence v meziprocesorové komunikaci (100 mikrosekund až jednotky ms)
- Typický výpočetní model: **zasílání zpráv**

Gridy

- Systém distribuovaný po geograficky rozsáhlých prostorech (země, kontinent, …)
- Propojeny samostatné počítače (včetně paralelních)
- Propojení počítačů WAN sítí
- Extrémně vysoká latence v meziprocesorové komunikaci (desítky až stovky ms)
- Prakticky jediný výpočetní model: **zasílání zpráv**

2. přednáška

Programové vybavení

- Nadstavba technických prostředků – problém např. sdílení – aby někdo nelezl někomu jinému do soukromých dat a podobně
- Vrstvy operačního systému – OS vyvíjeny po desítky let
    - Technická vybavení
    - Správa paměti
    - Správa procesů
    - Správa periferií (I/O)
    - Správa souborů dat
    - Uživatelský prostor (nepřesné)

Programové vybavení – jiný pohled

- Operační systém
    - UNIX, Linux, OS/370, MS Windows
- Programovací jazyky
    - C, Pascal, Ada, Occam, ML, Prolog, Perl, Python, Java
- Podpůrné nástroje
    - Debuggery, profilery
- Aplikační programy – např. webové browsery, atd.

Programovací jazyky

- Rozlišujeme:
    - Styl
    - Míru abstrakce
    - „Dialekt“

Programovací jazyky – styl

- *Imperativní/procedurální*: C, Fortran (používá se např. na vědecké výpočty, vymyšlen v padesátých letech)
- *Objektově orientované*: Java, C++
- *Deklarativní/funkcionální* (vznik zhruba ve stejné době, jako ↑): ML, Lisp, MIRANDA
- *Deklarativní/logické* (Popis toho, co bude vstupem a co výstupem; celé zpracování se nechá na programu samotném): Prolog, GHC
- *S jediným přiřazením* (nelze použít x=x+1 – není jasné, jakou má x hodnotu, zda tu před přiřazením, nebo tu potom): SISAL
- *Produkční systémy* (najít výsledky, co splňují jisté podmínky, zřetězit je postupně za sebe; umožňuje i nějak vidět argumenty pro daný výsledek): OPS5
- *Sémantické sítě* (umožňuje používat vlastní jazyk, aby se člověk (vědec)nemusel soustředit na syntax): NETL
- *Neuronové sítě* (snadná konstrukce sítě neuronů, definice potřebných věcí): SAIC ANSpec

Programovací jazyky – míra abstrakce

- *Strojový jazyk*: přímo kódy jednotlivých instrukcí – přirozená nevýhoda: člověk se musí věnovat všem detailům, nevynechat jedinou instrukcí, nemůže pracovat se složitějšími datovými typy
- *Assembler* (pracuje stále na úrovni strojového jazyka; pokud se program přesune na jiný procesor, nemusí na něm fungovat a musí se přepsat): jména instrukcí (už umí instrukce pojmenovat), operandy, pojmenované cílové adresy skoků
- *Vyšší jazyky*: obecné konstrukce, tvoří „kontinuum“
    - Agregované datové typy (vektory, matice, struktury, data různého typu)
    - Cykly namísto skoků
    - Procedury a funkce (lišily se např. v tom, zda povolovaly rekurzi nebo ne)
    - Procesy a vlákna (nebo jiné nástroje pro synchronizaci procesů; na jakých úrovních pracuje se vstupy a výstupy)

Programovací jazyky – implementace

- **Překladače** – zpravidla má překlad dvě fáze
    - Zdrojový kód – mezijazyk (třeba Java bitecode) – strojový jazyk
    - Překlad a sestavení – překlad vytvoří program ve strojovém jazyce; přeloží program, který jsme mu předložili, ale ten mohl obsahovat spoustu funkcí, které nejsou přeložené a jsou na ně relativní odkazy. Úloha sestavení: v překladu se zanechají relativní odkazy k funkcím, teprve později se tyto odkazy převedou do absolutních
- **Interprety**
    - Abstraktní počítač
    - Vhodné pro složité operace (např. práce s texty, s maticemi a algebraickými objekty)
- **Just-in-time překladače** (nejen Java) – překládají se postupně jen části, které je potřeba vykonat, není třeba např. stále překládat vnitřek cyklu, stačí přeložit jednou
    - Známy již od osmdesátých let (řešil se tak nedostatek paměti)

Výpočetní model

- Souvislost mezi architekturou a jazyky
    - von Neumannova architektura a imperativní jazyky
    - objektová architektura a objektově orientované jazyky (např. Java virtual machine)
    - redukční architektura a funkcionální programování – virtuální stroje, které mají charakter optimálního počítače pro funkcionální programování.

Výpočetní model – varianty

- Datově orientovaný (nejpoužívanější) – výsledek: čísla
- Objektový – v.: množina objektů a jejich aktuální stav
- Funkcionální – podoba funkcí a funkčních závislostí – v.: číslo nebo funkce
- Logický – logické formule – v.: číslo, text nebo formule, logické závislosti

Aplikace

Dávají počítačům smysl – nejen na počítání, pronikly do všech oblastní našeho života

- Vědecko-technické výpočty (vojenství: atomová bomba) – primárně numerické, původně stavěny ze zvědavosti; simulace a ověření, zda experiment má cenu udělat, nebo ne (cena, nebezpečí)
- Zpracování informací (Hollerith/IBM: census USA – sčítání lidí za použití děrných štítků) – informační systémy nejrůznějšího druhu
- Zábava (počítačové hry, video-on-demand) – původně počítače stály mnoho peněz a byly ohromné, hry na nich byly nemyslitelné
- Řízení (management strojů i lidí, real-time systémy)

Číselné soustavy

- Definovány základem: desítková, dvojková, osmičková, šestnáctková
- Volně mezi sebou převoditelné (celá čísla bez ztráty přesnosti)
- Celá čísla a zlomky
- Reálná čísla
- Konečná reprezentace – nejsme schopni vyjádřit žádné číslo, které se donekonečna rozvíjí
- První počítače byly v desítkové soustavě – pracovaly s informacemi =finance, které vyjadřujeme v desítkové soustavě; eliminace problému se zaokrouhlením tím, že se počítače „naučily“ desítkově. Později se zaokrouhlování standardizovalo.

Dvojková soustava

- Základ číslo dvě:
  - Pouze dvě číslice/dva stavy
  - Vhodná pro reprezentaci v elektrických systémech
- Konečná reprezentace: interval hodnot
- Pro reálná čísla:
  - Rozlišitelnost (nejmenší zobrazitelné číslo): $X + \varepsilon > X$ a $X + \varepsilon/2 = X$
  - Přesnost (rozsah)
  - Zobrazení: mantisa *m* a exponent *e*
    $0 \leq m \leq 1$ && $x = m.2^e$
- Záporná čísla:
  - Přímý kód
  - Inverzní kód
  - Dvojkový doplňkový kód

Záporná čísla – zobrazení

- Přímý kód:
  - Přidáme znaménko
  - Dvě nuly: +0 a -0 (10 … 00)
  - Rozsah: <-MAX, -0> a <+0, +MAX>
- Inverzní kód:
  - Přidáme znaménko
  - Dvě nuly: +0 a -0 (11 … 11)
  - Rozsah: <-MAX, -0> a <+a, +MAX>

- Dvojkový doplňkový kód:
    - Inverze bitu a přičtení jedničky
    - Pouze jedna nula (11 … 11 je -1)
    - Nesymetrický rozsah: <-MAX -1, -1> a <+0, +MAX>
    - Skutečně používán v počítačích

Rozsahy čísel

- Podle počtu bitů:
    - Byte: 8 bitů, tj. <0, 255> nebo <-128, 127>
    - Půl slovo, 2 byte: 16 bitů, tj. přibližně $<-2.10^9, 2.10^9>$
    - Dvojslovo (nebo dlouhé slovo), 8 byte: 64 bitů, tj. přibližně $<-9.10^{18}, 9.10^{18}>$

Racionální čísla

- Formát dle IEE 754
- Součásti:
    - Znaménko
    - Mantisa (přímý kód, normalizace, m bitů)
    - Exponent (v kódu posunuté nuly, n bitů)
- Normalizace mantisy (exponent má n bitů):
    - Nejvyšší bit vždy jedna: 1.aaaaaa; 1 nezapisujeme
    - Nejmenší kladné číslo: $1.0 * 2^{-2\wedge(n-1)+1}$ (2^{-127} pro n=8)
    - Největší číslo: $1.0 * 2^{2\wedge(n-1)}$ (2^{128})
- Exponent (n bitů, dvojková soustava)
    - Přičteme 2^{n-1} -1 (=127 pro n=8), abychom získali správnou hodnotu pro uložení
    - 00000000 je -127
    - 11111111 je 128
- *Zvláštní a nenormalizovaná* čísla (třeba že nějaká posloupnost bitů je chápána jako ne-číslo)
- Rozsah zobrazení: <Největší záporné, Největší kladné>
- Přesnost zobrazení: počet bitů mantisy +1
- Rozlišitelnost: nejmenší nenulové číslo
    - Normalizované vs. Nenormalizované (2^mkrát menší, m počet bitů mantisy)

Jiné soustavy

- Osmičková
    - $001\ 101\ 101\ 111_2 = 1557_8 = 879_{10}$
- Šestnáctková
    - $0011\ 0110\ 1111_2 = 36F_{16} = 879_{10}$
- Používány především pro „hutný“ zápis binárních čísel

Operační systémy – trocha historie

- **Bootstrap loader** – první prazáklad OS. Z média, které je k dispozici, načte instrukce, které se budou provádět. Původně programátoři museli psát úplně vše, postupně tam začaly být programy, funkce atd., které byly obecně potřebné – základ OS.
- **Spooling** – počítač počítá a sám umí nahrávat další program (nahrají se do něj a on si je připraví) – sekvenčně umí spouštět nachystané programy. Funkce OS se posunula pro plynulé přecházení od programu k programu.
    - Nezávislé zavádění programu a jeho vykonávání
    - Vyžaduje DMA (*Direct Memory Access*)
    - Zavedlo multiprogramování
    - Stále zpracování dávek (*batch processing*)
- **Timesharing** – zpracování více interaktivních úloh. Umožňuje, aby jeden konkrétní fyzický počítač používalo více lidí naráz a jeho uživatelé se navzájem neomezovali. Tento počítač je musí vzájemně chránit – i vůči jim samotným (třeba pokud jeden z nich vyvíjí programy a dělá chyby).
    - Virtualizace počítače/CPU
    - Zpracování interaktivních úloh
    - Souvisí se zavedením disků (*Direct Access Storage Device*, DASD od IBM, 60‘ léta)

Operační systémy – účel

- Zkrásnění počítače:
    - Zjednodušení práce s počítačem
        - Práce s pamětí
        - Práce se soubory – požadavek na výrazně vyšší rychlost práce než třeba s terminálem
        - Přístup k periferiím
- Sdílení: zajistit sdílení vzácných zdrojů
- Musí zajistit:
    - Aby to vše fungovalo
    - Aby to fungovalo účinně (využití, propustnost, rychlost odezvy)
    - Aby to fungovalo správně
        - Omezení následků chyb (avšak pozor na chyby v samotném OS)
        - Oprávnění k přístupu (autentizace a autorizace) – a aby se toto dalo měnit

OS: Problém časování

- Periferie výrazně pomalejší než procesor
- Příklad
    - 1 GHz Pentium IV: 1.10^9 operací za sekundu
    - Běžný disk: 10ms pro přečtení 1 byte
    - Poměr: 1: 10 000 000
    - Stejné zpomalení člověka: 1 úhoz na klávesnici za cca 20 dní
- Možné řešení: prokládání I/O a výpočtu
    - Spusť diskovou operaci
      Prováděj instrukce nad jinými daty (alespoň 1M instrukcí)
      Počkej na dokončení

    - Příliš těžkopádné a složité
- Jiné řešení:
  Proces 1 {
  Spusť diskovou operaci
  Počkej na dokončení
  Zpracuj získaná data
  }
  Proces 2{
  Nějaká jiná operace
  }
    - Přehlednější
    - OS musí „přepínat" mezi procesy (priorita)

OS: Paměť

- Většina paměti nevyužita – i tam, kde je náš vlastní program
    - Zpracování cyklu (zbytek programu) – nebýt cyklu, musí mít program zhruba 1mld operací, aby proces trval 1 vteřinu
    - Zpracování konkrétních dat (ostatní neaktivní)
    - Čekání na I/O
- Virtualizace paměti
    - Data a programy na disku
    - Do paměti na žádost
    - Umožňuje
        - Každý program má „celou" paměť
        - Program může adresovat více jak rozsah fyzické paměti
- Ochrana paměti

OS: Základní složky

- Procesy a jejich správa
- Paměť a její správa
- Periferie a jejich správa
- Systém souborů
- Ochrana a bezpečnost

Procesy

- Proces je abstrakce průchodu programem – množina všeho, co tvoří běh konkrétního programu. Program může spustit libovolný počet procesů
    - Sekvenční model: program = 1 proces
    - Paralelní model: program > 1 proces
- Proces má interní stav, charakterizovaný
    - Programovým čítačem (program counter)
    - Zásobníkem (volání funkcí a procedur)
    - Vlastní pamětí pro data

Typy procesů – vzájemně se liší prací s daty

- **Klasické (heavy-weight) procesy** (např. UNIX)
  - Všechna data privátní
  - Sdílen pouze program (read-only)
- **Lehké (light-weight) procesy** či **Vlákna (threads)** – vlákna mají pouze minimum paměti, pracují se sdílenými proměnnými. Výhoda – umí sdílet data přirozeněji, zatěžují svou režií (vytvořením, zrušením, plánováním procesu) OS mnohem méně, než heavy-weight procesy
  - Minimum vlastní paměti
  - Většina dat sdílena

Procesy detailněji

- Vytvoření procesu – proces vytvoří nějaký předcházející proces
  - fork() a jeho varianty
  - Přesná kopie původního procesu
  - Rodič a potomek – potomek přepíše svůj kód, ale „obálku“ má od rodiče
  - První proces v OS vytvářen jinak (init v Unixu)
- Stavy
  - Start/vytvoření, připraven (ready) – pominuly důvody, proč byl zablokován, proces může být zase zpracováván na procesoru a čeká, až mu OS nějaký procesor přiřadí; běží (running), je blokován (čeká), skončil – zjišťování, zda skončil korektně, byl ukončen, atd.; úloha OS – plynulý přechod mezi těmito stavy, zajišťuje, aby se proces nezdržoval ve stavu „ready“ příliš dlouho (výjimkou jsou velmi nízké priority) – když se proces dostane do stavu „ready“, měl by se prakticky okamžitě spustit

Synchronizace – problém

- **Race condition:** soupeření v čase – aby nenastala situace popsaná dole. Kdo dřív přijde, ten dřív mele
  - Proces P {
    Load RegistrA, X Nahraj do RegistruA proměnnou X
    Load RegistrB, Y
    Add RegistrA, RegistrB
    Store RegistrA, X # X+=Y
    }
- Dvě instance procesu P, používají stejná X a Y – pustí se jako dva lehké procesy, naráz – po opakovaném spuštění dostaneme jiné výsledky. Proč? První proces začne, nahraje do registru A hodnotu X. Potom stále běží tento proces a nahraje do Registru B Y. V tomto okamžiku bude výsledek 2; tento proces doběhne, potom se pustí druhý proces, ale X už je 2 – výsledek tohoto druhého procesu už bude 3, ne 2.
  Co ale když pustí první, nahraje do A i B jedničku, potom proces zastaví, spustí druhý proces, ten také nahraje do A i B k sobě jedničku, potom OS zastaví tento proces a spustí opět první, nechá ho proces dopočítat – nyní bude celkový výsledek jiný (2).
- Nedefinovatelné výsledky
  - Je-li na začátku X=Y=1, pak na konci může být X=2 nebo X=3

Synchronizace – řešení

- Kritická sekce
  - Semafory: celočíselné proměnné (čítače)
  - Monitory: vyšší konstrukty programovacího jazyka
    Je možné semafor implementovat pomocí monitoru a naopak
- Smrtelné objetí (**deadlock**)
  - Deadlock (česky také uváznutí, vzájemné čekání) je odborný výraz pro situaci, kdy úspěšné dokončení první akce je podmíněno předchozím dokončením druhé akce, přičemž druhá akce může být dokončena až po dokončení první akce. Vzniká paradox, často označovaný jako 'Co bylo dříve? Slepice nebo vejce?'.
    V počítači se jedná o zablokování procesů (případně vláken) způsobené křížovým čekáním na synchronizačních primitivech. K uváznutí dochází v důsledku chyby programu nebo není uváznutí v programu úmyslně řešeno, protože řešení by bylo příliš náročné. Pokud v takovém případě dojde k uváznutí, je nutný zásah uživatele, který může násilně ukončit jeden z procesů nebo v případě práce s databází může jednu transakci zrušit (příkazem rollback).
- Odstranění sdílených zdrojů: zasílání zpráv
  - Synchronizace na úrovni zasílání a přijímání zpráv
  - Buffery

Procesy – plánování

- Sdílení (timesharing)
  - Časové kvantum
  - Přerušení
- Prioritní
  - Statistické
  - Real-time
- Plánovač (scheduler)

Správa paměti

- Dvě základní operace:
  - Alokuj/přiděl paměť (velikost, vrací počáteční adresu)
  - Dealokuj/uvolni paměť (velikost a počáteční adresa)
  - Většinou závislé (lze uvolnit jen přesně totéž, co jsme alokovali dříve)
  - Doplňková operace: změň rozsah alokované paměti (reallocate)
- Organizace paměti
- Čištění paměti (garbage collection)

Správa paměti OS

- Virtualizace paměti – nutno uvolnit fyzickou paměť
- Swapping
  - Celých procesů
  - „Děr“ v paměti
- Stránkování
- Segmentace

3 - 4. přednáška

Návrh OS – principy

- Efektivita
- Robustnost
- Flexibilita
- Přenositelnost
- Kompatibilita
- Bezpečnost

Efektivita

- Maximální využití dostupných zdrojů (sdílení mezi více procesy, uživateli, …)
- Použití jednoduchých a jasných principů
- Dekompozice návrhu
    - Objektově orientovaný návrh (pozor na přílišnou fragmentaci) – správa procesů, periferií, souborů, … Trik je v tom, jak moc velké tyto objekty budou – zda velké a složité, nebo jednoduché malé objekty, ale bude jich triliarda – problém se skládáním dohromady
    - **Agenti** – kooperativní skupina programů/procesů – nějakým způsobem spolu spolupracují a jejich interakcí se řeší nějaký problém, zefektivňuje práce
    - Komponentní programování – „další generace objektově orientovaného návrhu“ – jednotlivé návrhy nemusí být objekty (paměť), mohou být složitější a mít více účelů, ale dají se snadno navrhnout.

Robustnost – pokud nějaký program začne počítat příliš intenzivně, je potlačen

- Schopnost úspěšně se vzpamatovat po výpadku
- Řešeno redundancí (standartní inženýrské řešení): snižuje ovšem pozorovanou efektivitu
    - První výzkum v ČR koncem 50. a začátkem 60. let (Ing. Svoboda – jeden ze tří AAA computer engineers z ČR = úplná špička)
    - Běžné trojnásobné jištění (např. řídící počítače atomových ponorek USA) – mechanismy, které pokud se vyhodnotí všude chybně (musí se shodovat všechny tři výsledky – vypočítávají je různé procesory), spustí se „panika“ - zrovna v případě atomové ponorky třeba uspání reaktoru
      Tzn. pokud se tři výsledky shodnou = v pořádku
      dva shodnou = v pořádku + varování
      ani jeden neshodne = run for your life
- V současné době zájem o self-healing programy

Flexibilita

- Možnost úpravy (adaptace) podle změněných potřeb – třeba problém s IPv4 vs. IPv6
- Často používána ve významu *rozšiřitelnost* (extensibilita)
    - Definuje a fixuje se rámec (framework)
    - Přidání nové složky bez změny rámce snadné
    - Případně hierarchie rámců (přidání či modifikace nového rámce)

Přenositelnost

- Úzce souvisí s operačními systémy – přelom zhruba v 70‘ letech: do té doby byl většinou hardware to, co se neměnilo, měnil se OS – nyní je to naopak (výměna HW, OS zůstává)
- Dodatečná abstrakce detailů
    - Virtuální „disk“ namísto konkrétního zařízení
    - Programy psány bez odkazů na speciální vlastnosti
- Využití standardů
- Opět možný rozpor s požadavkem efektivity

Kompatibilita

- Odstínění specifických detailů – dnes už víceméně není problém vyměnit např. grafickou kartu za jinou
- Využití standardů
- Efektivita?
    - Nemusí být negativně ovlivněna – pokud se zde schází vývoj SW i vývoj HW, tak z toho profitují všichni

Bezpečnost

- Cíl: Ochrana dat a majetku před krádeží, zneužitím, či poškozením při současném zachování přístupu vybraných uživatelů; kdo má oprávnění s něčím pracovat, aby jej ochrana nerušila, kdo oprávnění nemá, aby neměl žádné možnosti, jak škodit
- Problémy:
    - Větší nároky na správu systému – např. zadat heslo, pamatovat si ho, stojí to čas; zadat různá uživatelská oprávnění (abychom si třeba sami sobě nezakázali zápis do určitého souboru aj.)
    - Snižuje snadnost použití
    - Klade dodatečná omezení na uživatele (disciplina)
    - Srovnání: MS Windows 95 vs. MS Windows NT – první určen pro koncového majitele v době před připojením ze síť a přepokládalo se, že bezpečnost je zaručena tím, které ze svých dětí pustíte k PC; sice měli logovací obrazovku, ale byl to fake – pokud se tam nenapsalo heslo, tak se nic nestalo a systém nás pustil dále. Katastrofa nastala při připojení k internetu – v Americe měli na internet paušál, tzn. PC zůstávaly pořád zapnuté (a na síti) – kdokoliv se k nim kdykoliv mohl připojit. Reakce ze strany Microsoftu – Windows NT – určené pro servery a ne desktopy (přestřelený bezpečnostní model). Třeba existovaly unixové OS, které neměly normální admin účet, ale museli se zároveň nalogovat třeba dva lidi, kteří potom museli spustit specifickou posloupnost operací, aby se provedla důležitá změna.
      Z toho plyne, že všeho moc škodí (i moc bezpečnosti) – lidé to potom obejdou.

Externí požadavky (na funkcionalitu OS)

- Stejný (podobný) HW a různé priority
    - Server: např. stabilita, bezpečnost, propustnost – třeba menší robustnost, snadnost
    - Pracovní stanice: např. snadnost ovládání, rozumný výkon ve všech oblastech
    - Specializovaná grafická stanice: maximalizace grafického výkonu
    - Řídící systém: požadavky real-time, robustnost – aby se např. nespouštěly aktualizace

Klasifikace OS

- Monolitický
- Vrstvený
- Modulární
- Mikro-kernel

Monolitický OS

- Původní operační systémy (proprietární) – byl vyvíjen lidmi, kteří vyvíjeli daný HW – špičkoví programátoři té doby (IBM)
- Abstrakce nepoužívána příliš dovnitř – maximální používání vlastností daného HW, ale špatně aktualizovatelné – s výměnou HW se vyměnil i OS
- Nejasné rozlišení funkcí uvnitř operačního systému
- „Velké“, špatně rozšiřitelné, špatně udržovatelné
- Poplatné době pomalejšího vývoje HW a jeho vysoké ceny

Vrstvený OS

- Vrstvy odpovídají procesům správy:
    - Správa CPU
    - Správa paměti
    - Správa periferií
    - Správa systému souborů
- Lepší abstrakce – definovala se rozhraní a práce byla jednodušší
- Komunikace mezi vrstvami – zpomalení systému; pokud narůstal výkon HW, rozšiřovaly se vrstvy
- Počítá s tím, že jádrem všeho je procesor (ve středu cibule), další vrstvy se potom přidávaly později – paměť, periferie, …

Modulární OS

- Moduly namísto vrstev – bylo by super, kdyby třeba správa systému souborů uměla pracovat se správou paměti (tzn. zvenku dovnitř cibule); místo vrstev větší množství modulů, které dohromady tvoří OS
- Zapouzdření (enkapsulace) funkcí
- Komunikace mezi moduly
- Příbuzný objektovému přístupu
- Lepší údržba – dají se zde opravit pouze jednotlivé moduly a na rozdíl od monolitického OS nemusí vědět jeden člověk vše
- Riziko vzniku „*fatware*“ – neskonale obrovský OS, ze kterého potřebujeme jen trošku – cokoliv nového je potřeba, napíše se na to další modul. Do jisté míry to chceme – zvýšení flexibility – ale je nutné vyvážení
- Tento způsob v podstatě využíván dodnes

Kernel operačního systému

- **Kernel**, též jádro operačního systému: - množina modulů, které v každém OS být musí
  - Základní složka OS
  - Odpovídá za:
    - Alokaci a správu zdrojů
    - Přímé ovládání HW (nízkoúrovňové interfaces)
    - Bezpečnost
- **Mikrokernel**:
  - *Malé je pěkné* – zkusme ty moduly ořezat ještě více, abychom maximalizovali možnost, že budou bezchybné (čím větší, tím větší možnost chyby)
  - Modulární přístup, malé moduly odpovídající za konkrétní operace
  - Řada funkcí až v uživatelském prostoru
  - Vysoce flexibilní, upravení operačního systému podle potřeby

Aplikační programová rozhraní (API)

- Definují způsob („calling conventions“) přístupu k OS a dalším službám
- Sestává se z definicí funkcí, datových struktur a tříd
- Představuje abstrakci volané služby
- Účel:
  - Přenositelnost
  - Snadná správa kódu
- Další použití:
  - Překlad mezi službami vysoké a nízké úrovně
    - Převod typů/struktury parametrů
    - Převod mezi způsoby předávání parametrů (by-value a by-reference) – pracuje se s pointery, musí se dávat pozor, aby se předala hodnota
- Velké rozšíření s UNIXem; programujeme malé moduly a ty potom vzájemně propojujeme podle toho, co je právě potřeba počítat. Pro vývojáře aplikací znamená API způsob, jak aplikace komunikuje s OS. Např. potřebuji zvýšit procesu prioritu – skrze API a zavolání abstrakci dané funkce se to provede

API – příklady

- Práce se soubory:
  - Otevření: int open(char *path, int oflag, …)
  - Čtení: int read(int fildes, char *buf, unsigned nbytes)
  - Zápis: int write(int fildes, char *buf, unsigned notes)
  - Zavření int close(int fildes)
- Práce s pamětí:
  - Alokace paměti: void *malloc(size_t size)
  - Uvolnění paměti: void free(void *ptr)
  - Změna alokace: void *realloc(void *ptr, size_t size)

Periferie z pohledu (modulárního) OS

- Zpřístupněny prostřednictvím příslušného API
- Abstrakce: možnost výměny konkrétního zařízení (disk, síťová karta) bez vlivu na způsob použití – záleží třeba i na různých úrovních abstrakce. Třeba jakým způsobem se pohybuje myš – po rastrech? Nebo má více tlačítek – jednotlivým tlačítkům můžeme přisuzovat různé funkce. Např. pro kolečko myši zase musí být jiná abstrakce – točí se kolečko nahoru, nebo dolů? Točí se vůbec? → Na úrovni abstrakce je potřeba mít co nejvyšší přenositelnost, ale na druhou stranu musíme umět rozpoznat speciální zařízení a přiřadit mu správné funkce
- Příznaky a klíče pro ovládání specifických vlastností: přenositelnost vs. efektivita
- Ovladače na nejnižší úrovni („Nejblíže hardware“)
    - Specifické „jazyky“ ovládání periferií na této úrovni
    - Práce se signály (např. změna stavu periferie)
- Začlenění ovladače do jádra
    - Kooperativní vs. hierarchické (možnost preempce) – zpracovává se proces a najednou přijde jiný, který řekne, že potřebuje proběhnout hned – co se stane?
    - Efektivita vs. stabilita
    - Formální verifikace ovladačů: Microsoft Static Driver Verifier
- Příklady
    - Práce s diskem
    - Ovládání klávesnice a myši (čtení signálů)
    - Grafika a ovládání grafických rozhraní
    - Síťové karty

Přerušení

- Operační systémy obecně reagují na asynchronní události (events) – asynchronní události přicházejí zcela nečekaně (třeba po síti) – OS to potom musí nějak vyřešit, nějak zareagovat
- Přerušení: mechanismus, jak přerušit vykonávanou práci na základě externí příčiny (nějaké události) – např. vstup/výstup: uživatel zmáčkne klávesnici a očekává nějakou reakci bez ohledu na to, co se tam právě děje

Význam přerušení

- Podpora I/O
- Problém v programovém vybavení
    - Neautorizovaný přístup
    - Nelegální instrukce nebo operandy – např. pokud píšeme vlastní program a začneme dělat blbosti, tak by to měl OS zarazit
- Požadavek počítačem řízeného systému – např. přehřívání
- Zásah operátora
- Výpadek HW

Příklady

- Přerušení od časovače (přeplánování procesů, timeout, …) – dá se časový limit (třeba 100ms) – po této době se proces přeruší
- Přerušení od periferie (klávesnice, myš, síťová karta, …)

- Přerušení z procesoru (dělení nulou, chybná operace, …)

Principy přerušení

- Přeruší běh aktuálního programu
  - Nutno uložit stav a zapamatovat si místo návratu – má ho jen přerušit, ne ho zabít
- Více zdrojů a příčin přerušení
  - Nutno rozlišit typy (příčinu) přerušení – musí zvládnout např. pokud přerušení přijde více než jedno
  - Nutno zapamatovat zdroj přerušení

Obsluha přerušení

- Obsluha přerušení realizována v kernelu – důvod, proč se to dělá v jádru OS:
  - Zajištění *serializace* – když přijdou dvě přerušení – zpracovávám jedno, přijde druhé – co teď? Potřebuji hierarchii?
  - Bezpečnost
- Vyvolá tzv. přepnutí kontextu (více přerušení)

5. přednáška

Další vlastnosti

- **Maskování přerušení** – jedna ze základních vlastností. I přesto, že přijde signál k přerušení, tak se přerušení neprovede
  - Dočasné a trvalé – dočasné: pokud už je jedno přerušení, tak po dobu, co jej zpracovává, neprovede další. Mohl by se totiž zacyklit. Trvalé – třeba ignorování dělení nulou.
  - Možná ztráta přerušení/události – některý ze signálu se může „ztratit“, systém si jej nezapamatuje. Případně se projeví až po tom, co maskování skončí.
- **Priorita přerušení**/obsluhy
  - Tři základní úrovně
    - Nemaskovaná přerušení: vyšší priorita
    - Aktuálně zpracovávané přerušení – pokud přijde přerušení se stejnou úrovní priority, s největší pravděpodobností bude zamaskováno
    - Maskovaná přerušení: nižší priorita

Polling

- **Polling** = opakované dotazování (na stav/událost) – alternativní možnost k přerušení. Může se použít oboje.
- Možná alternativa pro některá přerušení
  - Zaměstnává procesor – zatímco přerušení je externí signál, kdy procesor dělá, co chce a procesor reaguje až tehdy, pokud přijde signál, polling zaměstnává procesor stále („Přišel už ten signál?“)
  - Může zůstat v uživatelském prostoru – nevyvolává změnu kontextu

Systém souborů

- Základní funkce: - komponenta, která pracuje s entitou, které říkáme soubor
  - Vytvoření souboru
  - Čtení a psaní z/do souboru
  - Odstranění (smazání) souboru
  - Spuštění souboru (soubor = program) – speciální funkce z pohledu práce – přečtení komponentou OS, která umí spouštět procesy. Můžeme mít systém souborů na úrovni uživatelského procesu, ale nemůžeme garantovat, že si to majitel *nemůže* přečíst. Když Kernelu řekneme, že si ten daný soubor nikdo nemůže přečíst, tak se o to postará.
- Podpora na úrovni OS

Struktura systému souborů

- Hierarchické systémy:
  - Kořen (root)
  - Adresáře jako speciální typ (meta)souboru: drží informace o souborech, nikoliv jejich vlastní data – adresář je také soubor, ale prvky tohoto souboru jsou v podstatě informace o ostatních prvcích
- Databázové systémy: - lepší vyhledávání
  - Soubory (nebo jejich části) jako položka v databázi
  - Bohatší množina operací
  - Složitější implementace

Struktura souborů

- Dříve index soubory, které v sobě obsahovaly klíč – uživatel řekl: „Chci si přečíst údaje o souboru, který obsahuje blabla.“ Nevěděl, jak jsou v něm byty řazeny, o to se postaral OS. Ukázalo se jako neflexibilní, pokud OS chápe soubor jen jako posloupnost bytů.
- Posloupnost bytů – vnitřní struktura pro OS neznáma
- Posloupnost záznamů (records)
- Strom – každý uzel má vlastní klíč

Typ a přístup

- Typy souborů (v UNIXovém OS) – Windows má základní principy, jen třeba jinak pojmenované
  - Řádné: běžné soubory
  - Adresáře: udržení hierarchické struktury
  - Speciální: přístup ke konkrétnímu zařízení (/dev/mouse, /dev/audio, /dev/lp); speciální /proc systém - ve skutečnosti to jsou virtuální rozhraní k periferiím. Výhoda – stejnými operacemi (čtení/psaní) se pracuje i s periferiemi (otázka – jaký smysl má zápis do myši? Leda by třeba měla kolečka a uživateli někam ujela).
  - Blokové: náhodný přístup na základní úrovni (/dev/hd, /hd/kmem) - periferie, které nejsou organizovány po bytech, ale blokových jednotkách. Např. proto lze pracovat s debuggery – sáhnou si rovnou na dané místo v paměti a nemusí to číst přes náš program, který může mít chybu.

- Přístupové metody; příklady:
    - Sekvenční
    - Náhodný (random)
    - Indexsekvenční (není v běžném UNIXu)

Struktura na disku

- Možné typy – Soubor je souvislost bytů. Jak jsou tyto byty na disku uloženy?
    - Souvislé
        - Souvislá posloupnost bloků (složitá alokace, plýtvání místem)
    - Provázaný seznam
        - Každý blok odkazuje na další (může růst, vyšší režie – pro ukazatel, složitý náhodný přístup) – hodí se třeba pro soubory, u kterých víme, že se nebudou měnit. Pokud se často mění, tvoří se díry – ty sice jdou opět zaplnit, ale pokud potom chceme u souboru najít třeba n-tý byte, je potřeba sekvenční hledání celé paměti.
    - Indexové
        - Např. FAT (File Allocation Table) v MS DOSu
        - Tabulka pro všechny bloky na disku
        - Provázány odkazem na další blok daného souboru – na začátku je „tabulka“ a každá položka indexu odpovídá jednomu z bloků. Sekvenčně stačí projít pouze tuto tabulku. Tato tabulka se zkopíruje do paměti – rychlejší práce. Je výhodnější si držet tabulku i data co nejdéle v paměti – kdyby se to ukládalo zpět na disk pokaždé, když se provede změna, velice by to zpomalilo PC. Pokud ale vypadne proud, může se stát, že jsou na disku zapsána rozdílná data, než jsou v tabulce. Ovšem zásadní nevýhody: plochá organizace – pokud jsou větší soubory, je tam plno řetězení. Pokud se tabulka smaže, tak máme smůlu – je těžké se v tom potom zorientovat (postupné zaplňování děr, …).
    - Inodes – Tabulka pro každý soubor. Tyto tabulky jsou navrženy tak, že počítají s tím, že jsou soubory rozdělené na bloky (obvykle 512B). V jedné této tabulce jsou odkazy na další tabulky (4), které jsou v případě malých souborů prázdné. Pokud je soubor moc velký, tak se alokují další podřazené bloky. Teoreticky takto jdou udělat nekonečné soubory. Pomocí tohoto lze snadno najít n-tý byte.

Struktura – inodes

- Podobné indexovému
- Pevná délka tabulky pro každý soubor
    - Kratší soubory adresovány přímo
    - Pro delší soubory alokována další tabulka
    - Tabulky provázány hierarchicky (1., 2. a 3. úroveň)
- Flexibilní, malá režie

Volné bloky

- V tabulce
- Bitový vektor – je potřeba najít posloupnost volných bloků o určité délce
- Provázaný seznam – bloky na sebe vzájemně ukazují. Flexibilní operace.
- Většinou zpracovávány podle FCFS (First Come First Served)

Vyrovnávací paměť- cache

- Obecně přístup pro skrytí zpoždění (latence) – práce s komponentami, které mají rozdílnou rychlost. Nám se zdá, že pracujeme s tou, která je rychlejší.
- Nejčastěji používané bloky/soubory uloženy v paměti
- Pouze pro čtení (snazší) nebo i pro zápis
- Problém: konzistence při přístupech/zápisech z více míst
- Základní typy - My máme soubor pro čtení, ale co se stane, když do toho někdo jiný zapíše? Zápis se provede v cache; kdy uděláme změnu na disku?
    - **Write-through**: okamžitě po zápisu i na disk
    - **Write-back**: až po určité době (30s) – akceptujeme, že po nějakou dobu jsou data nekonzistentní. Jednou za čas (v UNIX-like systémech 30s) se data naráz zapíší. Když dojde k výpadku, tak si náš program myslí, že data zapsal. Tyto rozdílné přístupy se každý uplatní někde jinde.

Žurnálování

- Pokud dojde k výpadku proudu – **žurnálování (logování).** Pokud se zapisují data na disk, zapisují se i informace o tom, co jsme udělali.
  Žurnál je pro ochranu prováděné transakce využíván následujícím způsobem:
    - do žurnálu je zapsáno, co a kde se bude měnit
    - je provedena vlastní série změn
    - do žurnálu je zapsáno, že operace byla úspěšně dokončena
    - záznam v žurnálu je zrušen
- Pokud dojde v kterémkoliv okamžiku k přerušení, je možné pomocí dat uvedených v žurnálu uvést systém souborů do konzistentního stavu buď návratem zpět ke stavu před započetím transakce, nebo dokončením přerušené transakce.

Ochrana a bezpečnost

- Obecná ohrožení:
    - Přístup (čtení)
        - Nezanechává přímo stopy – Když si někdo přečte naše data
    - Zápis (modifikace)
        - Následné využití útočníkem modifikovaných dat
        - Zahrnuje i smazání/přepsání (chrání nás před námi – blbý program) – nebo pokud útočník jen pozmění náš program

        - Znepřístupnění služby (denial of service) – někdo nám znemožní používat systém, který chceme použít. Buďto vypnutím, nebo přetížením.
    - Možné útoky
        - Přihlášení, impersonifikace, … - ukradení identity, kterou se prokazujeme systému, že jsme to opravdu my. **Impersonifikace** – někdo se za nás začne vydávat.
        - Trojský kůň
        - Viry

Více o útocích

- Sociální inženýrství
    - Uhodnutí nebo získání hesla
    - Využívá důvěřivosti a naivity lidí
    - Technologie může pomoci jen do jisté úrovně
        - Nutnost koordinované shody dvou či více lidí – 7 klíčů ke korunovačním klenotům – např. nějaký systém z pre-Linuxové éry – mnohonásobné potvrzení identity pro provedení důležitého příkazu. Neosvědčilo se.
        - Kombinace fyzických nástrojů a tajemství (po krádeži karty je třeba ještě získat pin a naopak, samotný pin bez karty není k ničemu)
- Využití technických nedostatků
    - Bezpečnostní „díry“, „zadní vrátka“ apod.
    - Je možné minimalizovat korektními programátorskými praktikami a pravidelnou aplikací záplat (patchů)
    - Automatizované nástroje pro „oťukání“ systému
- **Botnety**
    - Sítě již napadených počítačů
    - Využitelné k dalším útokům

Principy návrhu bezpečnostních systémů

- Zveřejňování šifrovacích a souvisejících algoritmů
- Standardní nastavení = žádný přístup
    - Správce/uživatel musí aktivně rozhodnout, co komu dovolí
- Minimální oprávnění
- Pravidelné kontroly
    - „Díry“m nadbytečná oprávnění, …
- Jednoduchý a uniformní mechanismus
    - Složitost vede k nepochopení a to vede k chybám
- Úrovně oprávnění
    - Delegace oprávnění na konkrétní akci

Ochrana souborů

- Základní operace:
    - Čtení, zápis (včetně vytvoření), smazání, prodloužení a spuštění souboru
- Základní ochrana
    - Různá pro různé operace

- Specifikace, kdo smí co: ochranné domény
  - Skupina, která má stejná práva
  - Statické versus dynamické
  - Např.: já, moji přátelé, ostatní
  - POSIX (UNIX): user:group:other
  - Možná i jiná schémata

Řízení přístupu k souborům

- **Access Control List, ACL** (seznamy přístupových oprávnění)
  - Ke každému souboru je připojen seznam přístupových oprávnění
  - Sestává se z uspořádaných dvojic (doména, operace)
- Zjednodušená varianta (z UNIXových systémů)
  - Pouze tři záznamy: *u* uživatel, *g* skupina, *o* ostatní
  - Operace
    - r: čtení souboru (čtení obsahu adresáře)
    - w: zápis souboru (včetně vytvoření)
    - x: spuštění (sestoupení do podadresáře)
  - Příklad:
    - rw-r-----
    - Uživatel může číst i zapisovat, skupina smí jen číst, ostatní nesmí nic
- Plné ACL:
  - Libovolný počet záznamů
  - Více práv: smazání, změna, oprávnění
  - Negativní záznamy (explicitní odepření operace)
  - Dynamická dědičnost – propagace změn do podadresářů
  - Např. AFS (umožňuje vzdálený přístup a správu), Windows od verze 2000, ext4 s ACL
- **Capability List, CL**
  - Uspořádání podle domén, nikoliv podle souborů
  - Schopnost (capability) tj. práva přístupu patří procesu a ten je může:
    - Předávat dalším procesům (delegace)
    - Modifikovat (degradovat, nemůže rozšířit práva)
    - Smazat
  - Proces se při přístupu k souboru prokazuje odpovídající schopností
  - Možnost transferu schopností mezi procesy: vhodné pro distribuované systémy

Ochrana přístupu uvnitř POS

- Kernel a uživatelský prostor
- Oddělení na HW úrovni
- Každá stránka někomu patří
- Pouze kernel má přístup k HW
  - Kontroluje práva přístupu
  - Obsluhuje zařízení (pro všechny)
  - Garantuje serializaci přístupu
- Uživatelské procesy používají *volání* kernelu (jádra)
- Korektnost kernelu kritická – pokud se chová správně, pak je vše ok, pokud ne, tak ne

Přístup k paměti

- Příslušnost virtuálních stránek k procesu
- Výpadek stránky: nepovolený přístup
- Ochrana
    - Mezi procesem
    - Mezi procesy
    - Uvnitř procesu

Client-server model

- *Distribuované počítání* – pokud mám úlohu psanou na 1 procesor, tak i když dám dohromady 3 procesory, stále se to bude dělat jen na jednom; je potřeba úlohu pro více procesorů optimalizovat (třeba aby potřebovala čas 2-3x kratší)
    - Využití prvků (počítače) propojených počítačovou sítí
    - Dekompozice úlohy na podúlohy
    - Paralelní vykonávání podúloh
    - Na různých systémech propojených sítí
- Client-server model
    - Speciální případ distribuovaného počítání
    - Více strukturované
    - Asymetrické: *klient* posílá požadavek na zpracování *serveru*
    - Server pro jednoho klienta může být klientem pro jiný server

Vlastnosti modelu klient-server

- Klient a server samostatné procesy
- Na stejném nebo různých počítačích
- Interní informace je „soukromá“ pro každý proces
    - Klient i server se mohou vzájemně prokázat (autentizace)
- Komunikují duplexním protokolem
    - Komunikace může být šifrovaná

Požadované vlastnosti

- Interoperabilita
    - Klient a server mohou běžet na zcela odlišných systémech
- Portabilita
    - Stačí zajistit u klientů
- Integrace
- Transparence
    - Klient vidí jen „svůj“ server, nikoliv jeho další komunikaci (např. IS)
- Bezpečnost
    - Autentizace klienta i serveru
    - Šifrovaná komunikace
    - Důvěryhodný server – aby nedošlo k podvrhu

Příklady

- Telnet, ssh
- X Window systém na UNIXu
- Světová pavučina (WWW)
- Distribuované systémy souborů (AFS, NFS, Samba/CIFS)

Třívrstevný model klient-server architektury

- Základní rozčlenení
    - Data (server)
    - Logika
    - Prezentace (klient)
- Sousední možno kombinovat/rozdělit (tj. např. Logika může být součástí datové i prezentační vrstvy, a to i současně)

„Tlustý“ a „tenký“

- Platí pro server i klient, podstatné zejména v souvislosti s klienty
- **„Tlustý“ (fat) klient**
    - Značná spotřeba lokálních zdrojů (CPU, paměť, disk)
    - Komplexní provedení i instalace
    - Příklad: Mozilla
- „**Tenký“ (thin) klient:**
    - Jednodušší
    - Snadná správa a přenositelnost
    - Menší škálovatelnost (příliš mnoho práce dělá server)
    - Zpravidla vyšší nároky na propustnost sítě)

Middleware

- SW zajišťující funkcionalitu distribuovaných systému
    - „Zkratka“ v rámci protokolů
    - Stojí „nad“ OS, ale „pod“ aplikací
    - Propojuje oddělené komponenty distribuovaného systému
- Dovoluje aplikacím komunikaci přímo na vyšší abstraktní úrovni
- Realizuje jednu (RPC) nebo více (DCE) funkcí

7. přednáška

Obraz

Základní problémy

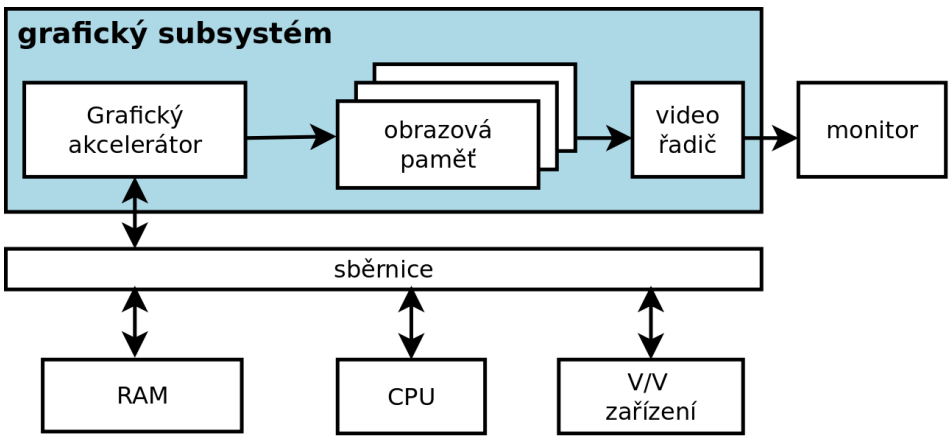
- Jak obraz vytvořit a reprezentovat
- Jak obraz manipulovat a zobrazit
- Jak napodobit realitu
- Jak to rychle vypočítat

Analýza a syntéza obrazu

- Analýza: obrazová data → modely Z obrazu si vytvoříme více či méně realistický model; další úvahy už děláme nad tímto modelem.
    - Extrakce informací z obrazových dat
    - Čárové kódy, detekce pohybu, identifikace objektů/osob, …
- Syntéza: modely → obrazová data
    - Vytvoření obrazových dat na základě datových modelů
    - Úprava fotografií, realistické zobrazení 3D scén, vizualizace dat (snaha grafickým způsobem zachytit nějaký soubor dat – třeba tabulkou)

Vizualizace

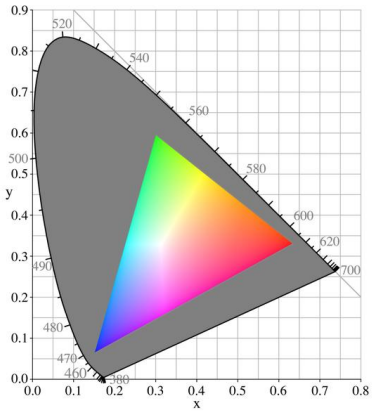
- Cíl: zobrazení komplexních dat uchopitelnou formou
    - Platí pro simulovaná data i výsledky měření
    - Snadnější průchod daty a jejich analýza
    - Usnadňuje porozumění objemným/komplexním datům
- Aplikace:
    - Výsledky vědeckých experimentů a simulací
    - Medicína
    - Strojírenství, defektoskopie (rentgen)



Fyzikální podstata barvy

- Světlo je elektromagnetické vlnění charakterizované vlnovou délkou a intenzitou
- Viditelné spektrum: 400nm (fialová) – 700nm (červená)

Barevné prostory



- Na každém zařízení lze zobrazit jen určité barvy
- Barevný gamut: množina barev, kterými dané zařízení disponuje
- Zde gamut typického CRT

Kódování barev

- **RGB**
  - Aditivní: sčítá barvy k bílé
  - Odpovídá skládání světla (např. LCD, CRT)
- **CMY**
  - Subtraktivní: odečítá barvy od bílé
  - Odpovídá míchání barev (např. inkoust)

Rastrový obraz

- Obraz je **2D pole pixelů** = obrazových bodů
- Barva každého pixelu je definována bity – tzv. barevná hloubka
  - 1 bit = černobílý obraz
  - Barevný obraz: 8, 15, 16, 24 (True color), až 96 bitů

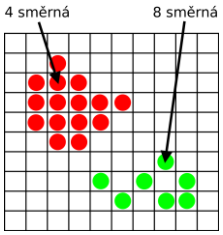
Technologie displejů

- CRT
  - Tři svazky elektronů jsou urychlovány a cíleny na lumiforovou vrstvu s RGB oblastmi
- LCD
  - Organické molekuly uložené mezi dvěma polarizačními filtry s kolmými osami polarizace
  - V klidové poloze polarizují světlo o 90° a umožňují jeho průchod
  - V excitované poloze nepolarizují a pixel se jeví jako nerozsvícený
  - Nevydává světlo: vyžaduje podsvícení či reflexní vrstvu
- Plazmové displeje
  - Plyn uzavřený v malých buňkách (3 na pixel) je excitován el. Polem a vydává UV záření
  - UV záření dopadá na fosfor uvnitř buňky a ten vydá viditelné světlo
- OLED
  - Několik vrstev organického materiálu uložených mezi anodou a katodou
  - Při průchodu el. proudu organickým materiálem dochází k emisi viditelného světla
  - Aktivní zdroj světla (nepotřebuje podsvícení), ohebné
- Dotykové displeje
  - Spojení obrazového výstupu a hmatového vstupu

Rastrová konverze úseček

- Cíl: převedení spojité úsečky do rastrové reprezentace
- Podél dané úsečky se v krocích po ose x počítá nejbližší pixel v ose y
    - Výpočet v pomocí round() v každém kroku je neefektivní
    - Inkrementální výpočet: *Bresenhamův algoritmus*

Výplň ploch

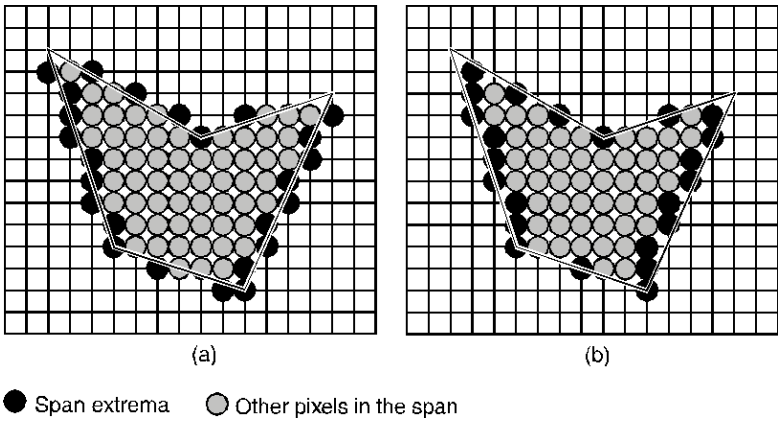


- Cíl: obarvení všech pixelů v dané oblasti
- Pixelově definované oblasti:
- Možné definice oblastí:
    - Všechny pixely dané barvy
    - Všechny pixely v dané vzdálenosti od pixelu
    - Oblast definovaná polygonem

Výplň polygonální oblasti

- **Záplavové** vyplňování
    - Zvol jeden pixel uprostřed oblasti
    - Rekurzivně obarvuj sousedy
- **Řádkové** vyplňování:
    - Rekurze probíhá po sousedících řádcích, ne axelech
    - Výrazně efektivnější
- **Paritní** vyplňování:
    - Najdi průsečíky řádky s polygonem
    - Seřaď podle polohy na ose X
    - Vybarvi sudé úseky

- Nejednoznačnost hranice a tedy i výplně:

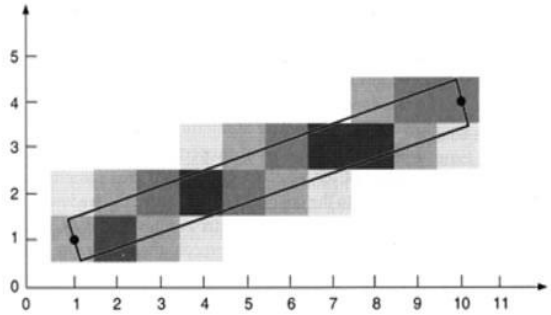


Anti-aliasing

- Převodem spojitého obrazu na diskrétní rastrovou reprezentaci vznikají chyby
  - Ztráta detailu
  - Vznik nežádoucích artefaktů
  - Rozpad tvaru
- Řešení:
  - Zvýšené rozlišení
  - Předfiltrování
  - Postfiltrování

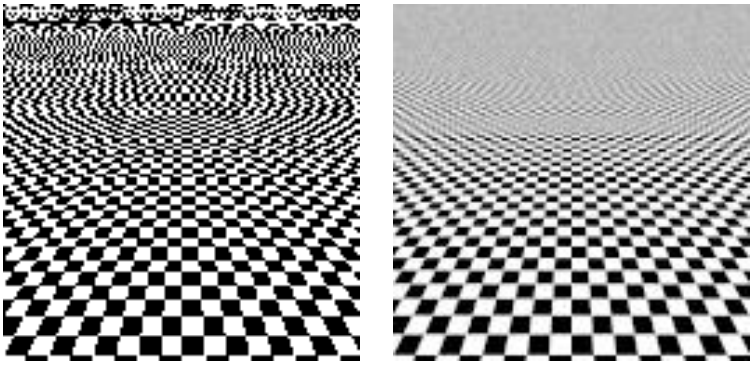
Předfiltrování

- Aplikuje se během rasterizace
- Každému pixelu je nastavena intenzita poměrně k velikosti plochy, kterou je zakrýván rasterovaným objektem



Alias: rozpad tvaru

- Zvýšené rozlišení (supersampling): obraz je vykreslen ve větším rozlišení a následně zmenšen



Rasterizace písma

- Bez anti-aliasingu: 

- Anti-aliasing: 

- Anti-aliasing a hinting 

    - Předpočítané parametry pro daný font a rozlišení

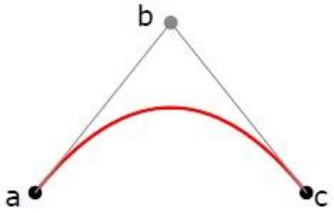
8. přednáška

Modely a modelování

- Cíl: popsat, „co je na obraze“
    - Ze základních primitiv se skládají komplexní tvary
    - 2D – vektorová grafika
        - Úsečka, křivka, elipsa/kružnice, mnohoúhelník, …
    - 3D – popis povrchů
        - 2D objekty s obsahem, parametrické plochy, spojování plátů, …

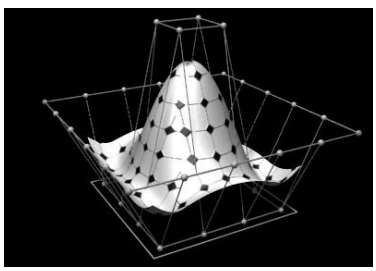
Parametrické křivky

- Úsečka
    - Koncové body a, b
    - $p(t) = (1 - t)*a + t*b$
- **Bezierova křivka**
    - Koncové body a, c
    - Řídící bod b
    - $p(t) = (1 - t)^2*a + 2t*(1 - t)*b + t^2c$

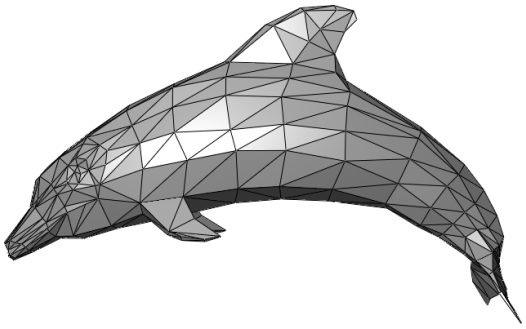


Parametrické plochy

- Umožňují popis hladkých zakřivených povrchů
- Vhodné pro průmyslový design
- Možnosti definice:
    - Okrajovými křivkami
    - Polygonovou sítí

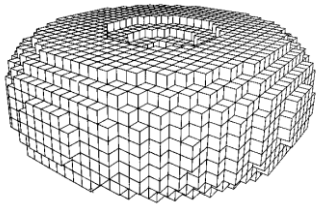


Polygonové modely



- Tvar je složen z konvexních 2D primitiv
- Dvoj-, troj-, mnohoúhelníky (polygony)
- Snadné vykreslení
- Pokud by se ale tento delfín rozřízl, tak bude vevnitř prázdný
- Techniky úprav povrchové sítě:
    - Tažení (extrudování) povrchu
    - Rotace profilu kolem osy
    - Zjemnění a deformace sítě
    - Konstruktivní geometrie těles
        - Komplexní objekty jsou vytvářeny pomocí boolských operací (sjednocení, průnik, rozdíl)

Objemové modelování



- Prostor uniformě rozdělen na voxely
- Voxely mají různou barvu, průsvitnost, …
    - **Voxel** = částice objemu představující hodnotu v mřížce 3D prostoru (analogie k pixelu v 2D)
- Aplikace:
    - Zobrazení medicínských dat - nejčastější
    - Objem zadáván po řezech
    - Možnost selektivního zobrazení

Renderování

- Cíl: vytvoření obrazu na základě modelu
- Popis scény:
    - Geometrie objektů
        - Polygonové/parametrické modely
        - Úroveň detailu a počet objektů ovlivňují výpočetní náročnost
    - Osvětlení
        - Popis zdrojů světla a jejich vlastností
        - Různé modely šíření světla
    - Textury
    - Směr pohledu
    - Stínování
        - Úprava úrovně jasu povrchu v závislosti na osvětlení
        - Pomocí stínů vnímáme hloubku, tvary, …

Textury

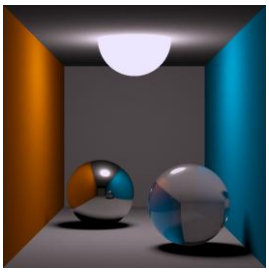
- Technika přidání detailu na povrch modelu
  - Určuje optické vlastnosti materiálu objektu
    - Barva, průsvitnost, lom světla, …
  - Přidává detailní změny geometrie
    - Normálové mapy (výstupky) – vytváří iluzi nerovnosti na povrchu úpravou normály v každém pixelu plochy. Modifikovaná normála potom ovlivní výpočet osvětlení plochy.
  - Na jeden povrch je možné aplikovat více textur
- Algoritmus nanášení textur se označuje pixel shader a realizuje jej GPU
- **Mapování** = jak se zakřivený objekt potahuje texturou
- Dva druhy:
  - **Rasterové**
    - Rasterový obraz je mapován („natažen“) na povrch
    - Výsledek je ovlivněn rozlišením textury a použitou interpolací (= nalezení přibližné hodnoty funkce v nějakém intervalu, je-li její hodnota známa jen v některých jiných bodech tohoto intervalu)
  - **Procedurální**
    - Vlastnosti pixelů povrchu jsou zadány funkcí
    - Vyžaduje programovatelný HW
    - Dobře škáluje na výsledné rozlišení obrazu
    - Výhodou je, že nezáleží na rozlišení, procedurální textura se přizpůsobí velikosti renderovaného obrazu. Nevýhodou ale je, že ne všechny povrchy lze matematicky vyjádřit.

Zpětné sledování paprsku (Raytracing)

- Sleduje cesty paprsků z oka do zdrojů světla
- Daný stupeň odrazů
- Umožňuje stínování, lesklé povrchy, …
- Metoda renderování (výpočtu a zobrazení) 3D počítačové grafiky; přesněji řečeno metoda globálního osvětlení. Na rozdíl od běžného života, kdy se paprsky pohybují od zdroje, odráží se a lámou, až se nakonec střetnou s okem pozorovatele, zde paprsky vycházejí z kamery. To protože ze zdrojů světla vychází nekonečné množství paprsků a nedalo by se v rozumném čase spočítat, které dopadnou na pixely plátna, přes které se oko dívá.

Distribuované sledování paprsku

- Každý paprsek nahrazen svazkem paprsků
- Výsledkem je průměr získaných hodnot
- Umožňuje hloubku ostrosti, měkké stíny, …



Renderování na GPU

- Renderování v reálném čase:
  - Nižší nároky na kvalitu, důležitý výkon (25 fps)
  - Rasterizace místo raytracingu
    - Geometrie transformována do 2D
    - Určeny viditelné trojúhelníky
    - Převedení na pole pixelů
- Programovatelné GPU
  - Novější generace GPU umožňují obecnější výpočty
  - Komplexnější per-pixel efekty (bump mapy, shadery)
  - Možné využít GPU i mimo jednoduchou rasterizaci (raytracing, konverze videa, vědecké výpočty)

Vývoj práce s GPU

- Statické API (application programming interface)
  - Pevně daná množina funkcí
  - Abstrakce od konkrétního HW
  - OpenGL 1.0, DirectX do verze 7
- Programovatelné shadery
  - **Shader** je počítačový program sloužící k řízení GPU (procesoru grafické karty).
  - Umožňují vytváření jednoduchých procedur vykonávaných na GPU
  - OpelGL 2.0, DirectX a výše
  - S příchodem OpenGL API a obdobných funkcí v DirectX se k možnostem GPU přidaly i programovatelné jednotky shader. Každý pixel či vertex může být zpracován krátkým programem předtím, než je zobrazen.
- **GPGPU** (General-purpose computing on graphics processing units)
  - Další rozšíření programovatelnosti GPU
  - Grafická karta jako stream procesor
    - Stream procesor - obecné označení výpočetní jednotek/procesorů pro paralelní zpracování dat - zpracovávají data souběžně (paralelně) s hlavním procesorem. V současnosti se setkáváme se Stream procesory jak u CPU, tak u GPU.
    - Streamování - proces, kdy datový tok není ukládán na disk, ale je rovnou zpracováván. Streamování se nejčastěji používá pro video a audio, zpracovává se tedy přehrávačem či přímo internetovým prohlížečem.
  - OpelCL, CUDA, …
  - Díky tomuto vlastně GPU zvládá operace, které normálně provádí pouze CPU

OpenGL

- OpenGL (Open Graphics Library) je průmyslový standard specifikující multiplatformní rozhraní (API) pro tvorbu aplikací počítačové grafiky. Používá se při tvorbě počítačových her, CAD programů, aplikací virtuální reality či vědeckotechnické vizualizace apod.
- Základní funkcí OpenGL je vykreslování do obrazového rámce (framebufferu). Umožňuje vykreslování různých základních primitiv (bodů, úseček, mnohoúhelníků a obdélníků pixelů) v několika různých režimech. V OpenGL se nepoužívá objektově orientované programování.

- Rozhraní OpenGL je založeno na architektuře klient-server – program (klient) vydává příkazy, které grafický adaptér (server) vykonává. Díky této architektuře je možné, aby program fyzicky běžel na jiném počítači než na tom, na kterém se příkazy vykonávají, a příkazy se předávaly prostřednictvím počítačové sítě.

DirectX

- Microsoft DirectX je v informatice sada knihoven poskytujících aplikační rozhraní (API) pro umožnění přímého ovládání moderního hardwaru. Jejich cílem je maximální využití možností hardware jak po stránce nabízených funkcí, tak z hlediska maximálního výkonu, což je využíváno pro tvorbu počítačových her, multimediálních aplikací i grafického uživatelského prostředí (viz Windows Aero).
- Původně měla API samostatná jména (a dodnes mají) například Direct3D, DirectDraw, DirectMusic a další. Název DirectX je tak zkratkou pro všechny tyto knihovny, kdy X nahrazuje název knihovny a později se tak stal i názvem celé kolekce. Když Microsoft začal vyvíjet vlastní herní konzoli, X bylo použito jako základ názvu (Xbox) a značilo tak, že i tato konzole poběží na technologii DirectX.
- Jak z názvu vyplývá, je DirectX produkt firmy Microsoft určeným výhradně pro operační systém Microsoft Windows.

9. přednáška

Počítačové sítě

- Skupina HW prostředků a počítačů propojených komunikačními kanály, které umožňují sdílení informací a zdrojů

Možné účely

- Komunikace uživatelů (přenos textu, řeči, videa, atd.)
- Sdílení HW zdrojů
- Sdílení dat a informací
- Poskytování SW služeb

Základní vlastnosti sítě

- Doručení dat (správnému příjemci)
- Správnost doručení (nepoškozená data)
- Včasnost doručení

Ideální vs. skutečné sítě

- **Ideální sítě** – přirovnání k matematice
    - Transparentní pro uživatele/aplikace
    - Neomezená propustnost
    - Žádné ztráty dat
    - Žádné zpoždění

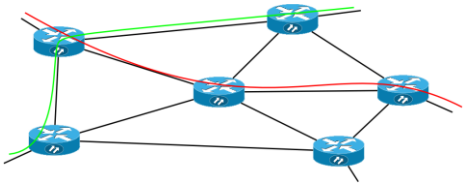
    - Zachovávají pořadí paketů
    - Data nemohou být poškozena
- **Skutečné sítě**
    - Mají vnitřní strukturu, která ovlivňuje doručení dat
    - Omezená propustnost – v akademické síti propustnost páteře Brno-Praha: 4x10GB
    - Dochází ke ztrátám dat – při zpracování dat přes aktivní prvky sítě a také při vlastním šíření signálu v médiu. Třeba na 100 000km s 5 aktivními prvky je latence asi 120ms. Síť navíc nezajišťuje, že všechny pakety půjdou přes stejné linky (a stejně dlouho).
    - Data se variabilně zpožďují
    - Pořadí paketů není garantováno
    - Data mohou být poškozena

Požadované vlastnosti

- Efektivita – efektivní/maximální využití dostupné přenosové kapacity
- Spravedlivost – stejná priorita přístupu všech uživatelů ke všem datovým tokům
- Decentralizovaná správa – síť se může rozdělit na dostatek domén tak, aby každá doména mohla mít vlastní správu a mohla komunikovat s ostatními doménami
- Rychlá adaptace na nový stav topologie sítě – změny topologie mohou být velice rychlé – jednak díky novým technologiím, druhak třeba při stavebních úpravách, příchodu nové firmy s vlastní sítí, nebo připojování studentů k síti, … Síť musí rychle zareagovat, adaptovat se a připojit nový uzel.
- Spolehlivost
- Řízení toku dat – ochrana proti zahlcení – rozdíl zahlcení sítě během dne, použití sítě se v čase mění (na univerzitě třeba kolaps sítě někdy v roce 1986)

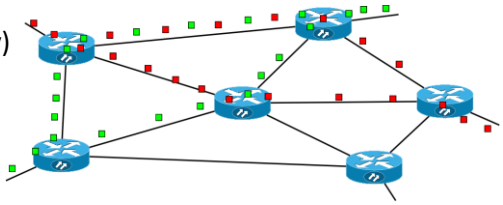
Základní přístupy

Spojované sítě (přepínání okruhů)



- Komunikace ve dvou fázích: navázání spojení → přenos dat
- Spojení (cesta sítí) je udržována během celé komunikace
- Síť udržuje stav – informace o vystavěných spojených
- „abstrakce drátu“
- Snadné zaručení kvality služby; co je to kvalita služby? Službu považujeme za kvalitní, jestliže jsme s jejími výsledky spokojeni. Má to jednu nevýhodu, nemůžeme ji přímo ovlivit
- Např. **analogové telefonní sítě** (s telefonistkami)
- Nevýhoda – špatná škálovatelost

Spojované sítě (přepínání paketů)



- Data rozdělena vysílající stranou na malé pakety (diagramy)
- Každý paket prochází sítí samostatně
- Pakety mohou přicházet v různém pořadí
- Přijímající strana pakety opět složí do původní podoby
- Není třeba uchovávat stav sítě, větší robustnost
- Velmi problematická implementace kvality služby – záleží na tom, kudy data putují
- Např. **Internet**

- Mnohem lepší škálování; neudržuje žádné spojení a aktivní prvky zpracovávají vše bez ohledu na to, odkud kam ta data jdou
- Jsou schopny mnohem většího růstu než spojované sítě. Dokonce i telefonní sítě kolikrát přenášejí data pomocí nespojované sítě

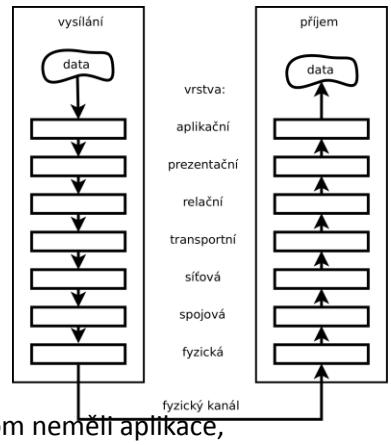
Komunikační protokoly

- Motivace:
  - Forma komunikace/domluvy musí být předem známa všem zúčastněným stranám
- Komunikační protokol:
  - Určuje, *co* je předmětem komunikace, *jak* daná komunikace probíhá a *kdy* probíhá
  - Definuje (zcela základní věc, nutno se naučit):
    - **Syntax** – strukturu/formát zasílaných dat
    - **Sémantiku** – funkční význam zasílaných dat (jak mají být interpretována)
    - **Časování** – kdy je třeba zaslat kterou zprávu
  - Např. TCP, UDP (← transportní), IP, IPv6 (← síťové), SSL, TLS, SNMP (← aplikační), HTTP, FTP, SSH, Aloha, CSMA/CD, …

ISO/OSI model

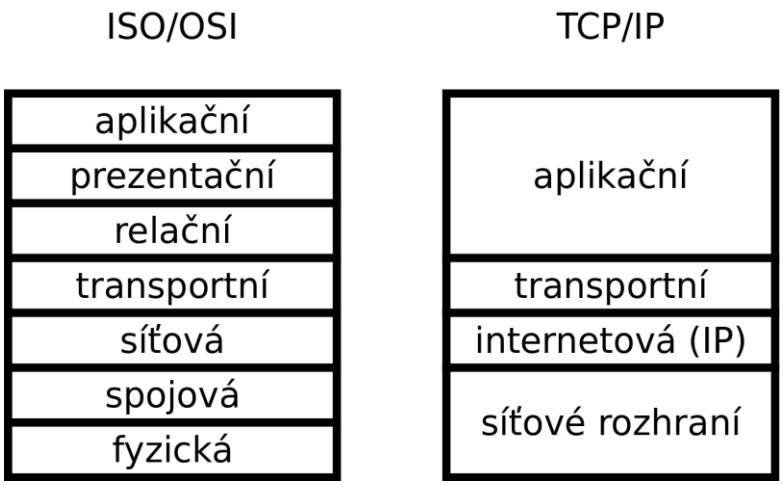
- 7 vrstev, každá odpovídá za konkrétní funkcionalitu
- Každá vrstva komunikuje pouze se sousedícími vrstvami
  - Využívá služeb nižších vrstev k poskytování svých služeb vyšším vrstvám
  - Funkcionalita je izolována v rámci vrstvy
  - Tento přístup je didaktický – dobrý na pochopení, jak co funguje; v reálu to není striktně oddělené, existují „zkratky“ mezi vrstvami
- Logicky se komunikace odehrává mezi stejnými vrstvami komunikujících stran, fyzicky prochází všemi nižšími vrstvami
- Vrstvy popisují pouze abstrakci, skutečné implementace se mohou odlišovat

Vrstvy:



- **Spojová (linková)** vrstva spojuje třeba dvě různé topologie sítě. Na síťové vrstvě získává každé zařízení svou adresu.
- **Transportní** - vlastní transport dat, zavádí spojení a další typ adres (port)
- **Relační** – zajišťuje relace
- **Prezentační** – jednoznačná interpretace dat mezi různými architekturami
- **Aplikační** – samotná aplikace; to, proč to celé má smysl. Kdybychom neměli aplikace, nemusíme mít sítě.

Srovnání s ISO/OSI



Síťová vrstva – IP

- Cíl: propojování lokálních sítí do velkých, komplexních sítí (internet)
- Internet protokol (IP):
  - Odpovídá za dopravu dat mezi uzly
  - Nespojovaná komunikace: základní jednotkou přenosu paket
  - Dnes máme dva IP protokoly **– IPv4 a IPv6**; původní vývojáři neodhadli rychlost, s jakou se síť rozšíří. IPv6 zavádí 128b adresu a považuje se za finální řešení problému s nedostatkem adres
  - Zajišťuje směřování paketů v síti (musí znát cílovou adresu; zná i výchozí)
  - Jednoznačná identifikace (adresace) každého zařízení
  - Metody základního monitoringu sítě

IP adresy

- Jednoznačné určení uzlu pomocí číselné adresy
- Adresa: 32 bitů (IPv6 128 bitů)
- Běžně se zapisuje dekadicky ve formátu: A.B.C.D
- Zleva hierarchická: 147.251.48.1
- Typy adres:
  - Individuální (**unicast**) – má je každé zařízení
  - **Broadcast** – slouží k zaslání dat všem uzlům v dané lokální síti (LAN) – není v IPv6, tam je nahrazen multicastem; je tam i nový typ (v tom IPv6) – **anycast** – data jsou doručována jednomu členu ze skupiny
  - Skupinové (**multicast**) – data směřována příjemcům, kteří o ně projevili zájem

Doménová jména

- IP adresy nevhodné pro lidi (IPv6 už vůbec)
- Jmenná služba: převod IP adres a jmen
- **DNS (Domain Name Service)**
  - Zprava hierarchická: aisa.fi.muni.cz
  - Lokální/kořenové DNS servery

IP protokol verze 6 (IPv6)

- Proč nový protokol?
  - Relativně rychlé vyčerpání adresního prostoru IPv4
  - Potřeba podpory aplikací reálného času, zabezpečení, autokonfigurace, mobility
- Vlastnosti:
  - Větší adresní prostor – 128 bitů (hexadecimální zápis)
  - Rozšiřitelný – rozšiřující hlavičky (hlavičky vypadají úplně jinak, než u IPv4)
  - Podpora přenosů v reálném čase – prioritizace provozu
  - Podpora zabezpečení přenosu – autentizace, šifrování provozu, …
  - Podpora mobility pomocí domácích agentů – u každého zařízení se předpokládá, že má někde svou domovskou síť. Pokud je mobilní, je mu v nových sítích přidělena vždy nová IP adresa. Směrovač v domovské síti je zde jako domácí agent a když mu zařízení pošle svou novou IP adresu, tak mu potom přeposílá data.
  - Podpora autokonfigurace

Přenos paketů

- Cíl: dopravit paket skrz síť od vysílače k přijímači
  - V podstatě grafový problém: síť jako graf
  - Algoritmy nalezení cesty mezi dvěma uzly grafu
  - Uzly znají cestu (explicitně, či implicitně)
  - Podle cílové adresy rozhodují uzly, kam poslat paket dále; není řešena cesta od začátku ke konci, ale krok za krokem; vnitřní uzly rozhodují, kam dále data půjdou. Tomu se říká **směrování**
- Dvě fáze směrování:
  - Nalezení směrovacích tabulek (musí se udržovat aktuální)
  - Vlastní zasílání datových paketů

Směrovací schémata

Tučně jsou označeny protokoly používané v internetu

- **Distribuované** / centralizované
  - Distribuované algoritmy – vzájemná kooperace uzlů při vytváření tabulek
  - Centralizované algoritmy – stav sítě se posílá do centra. Centrum spočte tabulky a zasílá je zpět uzlům
- **„Krok za krokem“** / zdrojové
- **Deterministické** / stochastické
- **Jednocestné** / vícecestné
- **Dynamické** / statické
  - Dynamické algoritmy – jsou robustní, tabulky jsou aktualizovány v reakci na změny v topologii
  - Statické algoritmy – jednorázové (často ruční) tabulky; jsou vhodné pro statickou topologii

Transportní vrstva

- Pojem **portu**
  - Více aplikací na jednom stroji: nutná jemnější adresace
  - Port: adresa poskytované služby
  - Síťový port je speciální číslo (1 až 65535), které slouží v počítačových sítích při komunikaci pomocí protokolů TCP a UDP k rozlišení aplikace v rámci počítače.

- **UDP** (User Datagram Protocol)
  - Pouze nezaručený přenos
  - Datagramy jsou zasílány do sítě bez další kontroly
    - Žádná garance ani kontrola doručení
  - Na rozdíl od protokolu TCP totiž nezaručuje, zda se přenášený datagram neztratí, zda se nezmění pořadí doručených datagramů nebo zda se některý datagram nedoručí vícekrát.
  - Protokol UDP je vhodný pro nasazení, které vyžaduje jednoduchost nebo pro aplikace pracující systémem otázka-odpověď (např. DNS, sdílení souborů v LAN). Jeho bezstavovost je užitečná pro servery, které obsluhují mnoho klientů nebo pro nasazení, kde se počítá se ztrátami datagramů a není vhodné, aby se ztrácel čas novým odesíláním (starých) nedoručených zpráv (např. VoIP, online hry).

- **TCP** (Transmition Control Protocol)
  - Staví „spojovanou“ službu nad IP
    - Iniciace spojení
    - Garantovaný přenos
  - Schopen reagovat na zahlcení (zpomalí a znovu postupně „přidává“)
  - Hlavní přenosový protokol na internetu
  - TCP ověří, že se pakety neztratily tím, že každému paketu přidělil pořadové číslo, které se také použije k ověření, že data byla přijata ve správném pořadí.
  - TCP modul na straně příjemce posílá zpět potvrzení pro pakety, které byly úspěšně přijaty. Pokud by se odesilateli potvrzení nevrátilo do rozumné doby (round-trip time, RTT), vypršel by odesilatelův časovač a (pravděpodobně ztracená) data by vyslal znovu.
  - TCP protokol ověřuje, zda přenesená data nebyla poškozena šumem tím, že před odesláním spočte *kontrolní součet*, uloží jej do odesílaného paketu a příjemce kontrolní součet vypočte znovu a ověří, že se shodují.

10. přednáška

Relační vrstva

- **Relace** (též dialog):
  - Spojení mezi dvěma koncovými účastníky na úrovni bezprostředně vyšší, než je vrstva transportní
  - *Analogie telefonního hovoru*
    - Je potřeba jej vytočit = analogie transportního spojení
    - Pak je možné jeho prostřednictvím vést hovor (=relaci) dvou účastníků
- Vztah relačního a transportního spoje:
  - Jedno transportní spojení může zajišťovat dvě nebo více po sobě jdoucích relací
  - Více transportních spojení může zajišťovat jednu relaci
    - Takže to není 1:1

Služby relační vrstvy

- Řízení dialogu mezi koncovými účastníky
  - Možnosti vedení dialogu
    - **Plně duplexní** (v terminologii RM ISO/OSI: Two-Way-Simultaneous – TWS)
    - **Poloduplexní** (Two-Way-Alternate – TWA)
    - **Simplexní** (One-Way)
  - *Poloduplexní* – Nelze vést současně data oběma směry, ale lze je vést nejdříve jedním směrem, potom tím druhým
  - Poloduplexní režim je řízen prostřednictvím mechanismu předávání pověření k přenosu dat (data token)
- **Synchronizace (též checkpointing)** – máme kontrolní body a v případě, že se něco stane, jsme schopni se k těmto kontrolním bodům vrátit zpátky a od nich pokračovat
  - Situace:
    - Příjemcem dat je počítač, který přijatá data tiskne na tiskárně
    - Dojde k dočasné poruše tiskárny (např. zaseknutý papír)
    - Příjemce může přijít o určitý objem dat, která jinak v pořádku přijal (tj. která byla transportní vrstvou bezchybně doručena) – je potřeba vrátit se „o kousek zpět“ a ztracená data přenést znovu
  - Řešeno mechanismem kontrolních bodů (synchronization points, checkpoints)
    - Příjemci umožňují, aby si na vysílajícím vyžádal návrat k zadanému kontrolnímu bodu (nové vysílání dat)
    - Zavedeny dva druhy kontrolních bodů – hlavní (major) a vedlejší (minor) – rozdělení dat na různé části
- Relační vrstva ISO/OSI není v TCP/IP modelu uplatněna – očekává se, že je to součást aplikace
  - TCP/IP nabízí pouze přenosové služby na úrovni transportní vrstvy
  - Potřebuje-li některá aplikace služby obecnějšího charakteru (a la relační vrstva), musí si je realizovat sama

- Příklady „protokolů relační vrstvy“:
    - SSL, Secure Sockets Layer
    - SDP, Sockets Direct Protocol
    - RPC, Remote Procedure Call Protocol
    - NetBIOS, Network Basic Input Output System
    - H.245, Call Control Protocol for Multimedia Communication
    - ASP, AppleTalk Session Protocol

Prezentační vrstva

- Opět v TCP/IP není a je ponecháno na aplikaci, aby si prezentační vrstvu ošetřila sama
- Hlavní úkol: konverze přenášených dat do jednotného formátu – ne všechny počítačové architektury chápou čísla stejně, tak potřebujeme nějakého prostředníka, který zajistí to, aby data byla interpretována a chápána správně
- Různé architektury se liší ve vnitřní reprezentaci dat (kódování znaků, čísel, atp.)
    - EBCDIC kód (střediskové počítače firmy IBM) vs. ASCII kód pro kódování znaků
    - Jedničkový doplňkový kód (CBC Cyber) vs. dvojkový doplňkový kód (většina ostatních PC) pro reprezentaci celých čísel
    - Little Endian (mikropočítače Intel, PDP-11) vs. Big Endian (počítače řady IBM 360/370, mikroprocesory firmy Motorola)
- Dosažení jednotné interpretace dat na obou komunikujících stranách – dvě možnosti:
    - Vzájemné přímé přizpůsobení stylu „každý s každým“ (v závislosti na komunikujícím partnerovi)
    - Převod do společného „**mezitvaru**“ – takto se to dělá
        - To také znamená, že když se objeví nějaká nová architektura s novým kódováním, tak se nemusí přizpůsobovat zbytek, jen ona, aby si vzájemně rozuměli
- Prezentační vrstva využívá společného mezitvaru
    - Pro popis přenášených dat využit jazyk **ASN.1** (Abstract Syntax Notation version 1)
    - Aplikace prezentační vrstvě předává data + jejich popis v jazyce ASN.1
    - Nutnost domluvy na vzájemném kontextu
        - Definuje, jaké struktury budou přenášeny a jaká bude jejich přenosová syntaxe
- Další možné služby prezentační vrstvy:
    - Šifrování
    - Komprese
- Příklady „protokolů prezentační vrstvy“:
    - AFP, Apple Filling Protocol
    - ASCII, American Standard Code for Information Interchange
    - EBCDIC, Extended Binary Coded Decimal Interchange Code
    - LPP, Lightweight Presentation Protocol
    - NDR, Network Data Representation
    - XDR, eXternal Data Representation
    - X.25 PAD, Packet Assembler/Disassembler Protocol

Aplikační vrstva

- Poskytuje služby pro uživatele:
  - Aplikační programy (aplikace) specifické pro požadovaný účel
    - Např. elektronická pošta, WWW, DNS, atd.
  - Aplikace = Hlavní smysl existence počítačových sítí, pro aplikace byly ty sítě postaveny
- Zahrnuje síťové aplikace/programy a aplikační protokoly:
  - Aplikační protokoly (http, SMTP, atd.) jsou součástí síťových aplikací (web, e-mail)
    - Nejedná se o aplikace samotné
    - Protokoly definují formu komunikace mezi komunikujícími aplikacemi
  - Aplikační protokoly definují:
    - Typy zpráv, které si aplikace předávají (request / response)
    - Syntaxi přenášených zpráv
    - Sémantiku přenášených zpráv (jednotlivých polí)
    - Pravidla, kdy a jak aplikace zprávy vysílají
    - Viz. z předchozího učiva – syntax, sémantika, časování

Vybrané síťové aplikace

- Jmenná služba – DNS
- World Wide Web – HTTP
- Elektronická pošta – SMTP
- Přenos souborů – FTP
- Multimediální přenosy – RTP, RTCP

Jmenná služba – DNS

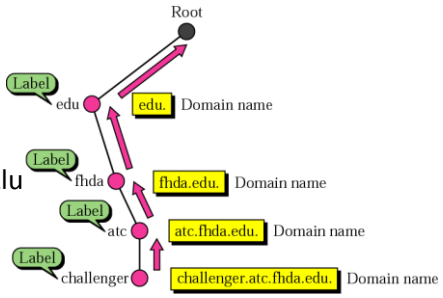
- **Domain Name System (DNS)** – služba pro překlad doménových jmen na IP adresy a zpět
  - Např. aisa.fi.muni.cz ←→ 147.251.48.1
- V začátcích Internetu řešeno za pomoci tzv. **host souborů**
  - Soubory s dvojicemi doménové jméno, IP adresa
  - Neškálovatelné řešení
    - S růstem Internetu nemožné mít tyto soubory (obsahující doménová jména celého Internetu) na každém uzlu
    - Navíc problém v nich vyhledávat a aktualizovat je

Jmenný prostor

- Jmenný prostor = způsob pojmenování předmětných entit
- 2 základní varianty:
  - **Plochý jmenný prostor** – jména bez jakékoliv vnitřní struktury
    - Např. mujRouterDomaVBrne
    - Hlavní nevýhoda: nemožnost využití ve velkém systému (nutnost centrální kontroly pro zamezení duplicit)
  - **Hierarchický jmenný prostor** – jména s hierarchickou vnitřní strukturou
    - Jména se sestávají z několika částí, každá s definovaným významem
    - Např. mujRouter.DomaVBrne.cz
    - Hlavní výhoda: možnost decentralizace správy (přidělování a kontroly) jmen-zodpovědnost vždy jen za určitou (pod)část doménového jména

Jmenný prostor Internetu

- **Doménový jmenný prostor (Domain Name Space)**
- Varianta hierarchického uspořádání (stromová struktura)
- Maximální počet úrovní: 128
- **Doména** – podstrom doménového jmenného prostorou
  - Jménem domény je doménové jméno jejího kořenového uzlu
    - 13 kořenových uzlů, z toho 11 v USA, 2 v Japonsku
- Každý uzel charakterizuje:
  - **Label** – řetězec (max. 63 znaků) popisující daný uzel
    - Jmenovka kořenového uzlu je prázdný řetězec
  - **Domain name** – sekvence jmenovek (oddělená znakem „.“) od daného uzlu ke kořenovému
    - Plné doménové jméno vždy končí znakem „.“

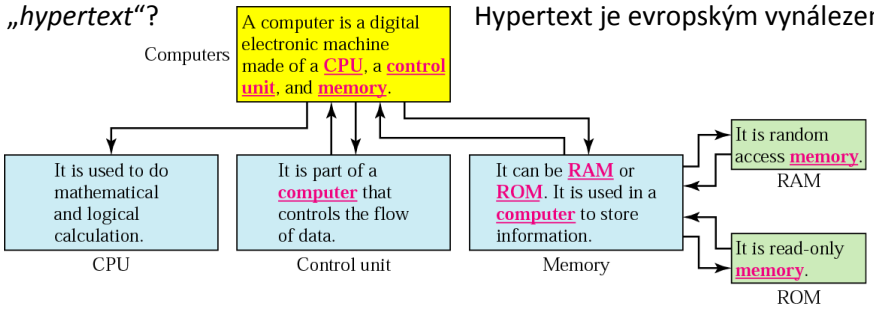


Domény v Internetu

- Jmenný prostor Internetu rozdělen na 3 základní typy domén:
  - **Základní (generic)** domény
    - com, edu, gov, int, mil, net, org, …
  - **Národní (country)** domény
    - Definují uzly podle příslušnosti ke státu
    - cz, sk, ca, us, jp, … - ca = kanada, jp = japonsko
  - **Reverzní (inverse)** domény
    - Slouží pro mapování IP adres na doménová jména
    - Struktura: převrácení IP adresa + identifikátor in-addr (inverze address) (IPv4) nebo ip6 (IPv6) + identifikátor arpa (z historických důvodů) → odkaz na ArpaNet

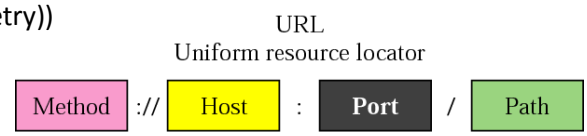
World Wide Web – HTTP

- *HyperText Transfer Protocol* (HTTP) – protokol pro přístup k datům na WWW
- Přenášená data mohou být ve formě textu, hypertextu, audia, videa, atp.
- Základní idea: klient vysílá požadavek, WWW server zasílá odpověď
  - Komunikace TCP protokolem na portu 80
    - Nemusí to být vždy na portu 80, třeba máme-li spuštěno více prohlížečů
- Co je to „*hypertext*“? Hypertext je evropským vynálezem (CERN)



URL – Uniform Resource Locator

- Součástí požadavku zasílaného na server je tzv. URL – to je ta adresa, kterou píšeme do prohlížečů
- Standardní mechanismus pro specifikaci „čehokoliv“ na Internetu
- Definuje zdroj, který chce klient získat
- Součástí URL je:
    - **Method** – metoda (protokol), který má být využit pro přístup k odkazovatelnému zdroji
    - **Host** – uzel, kde se odkazovaná informace nachází (kde má být vyhledána) – běžně využívané, zbytek často doplní sám prohlížeč
    - **Port** – volitelná součást, pokud je využit jiný než standardní port (80)
    - **Path** – cesta, kde se odkazovaná informace nachází (+ případně další informace (parametry))

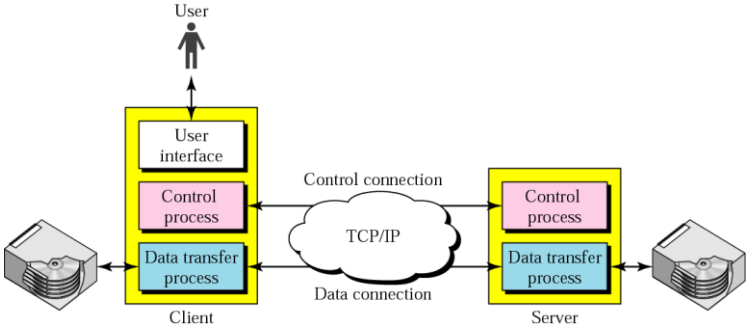


Dokumenty

- Základní kategorie WWW dokumentů:
    - **Statické** – na serveru uložené dokumenty s pevným obsahem
        - Např. HTML dokumenty
    - **Dynamické** – neexistují v předem definovaném formátu; jsou tvořeny webovým serverem dle požadavků klienta
        - Např. CGI skripty
    - **Aktivní** – serverem poskytnuté programy spouštěné na straně klienta
        - Např. Java aplikace

Přenos souborů – FTP

- *File Transfer Protocol* (FTP) – standardní mechanismus Internetu určený pro přenos souborů mezi uzly
- Oproti jiným klient-server aplikacím FTP klient s FTP serverem ustavuje **dvě samostatná TCP spojení**
    - Řídící zprávy zasílány tzv. *out-of-band*
        - **Řídící spojení** (TCP, port 21) – udržováno po celou dobu ustavení relace
        - **Datové spojení** (TCP, port 20) – otevíráno/zavíráno pro každý přenášený soubor



- **Řídící komunikace** – přenos požadavků klienta na jedné straně a odpovědí serveru druhým směrem
    - Domluva na parametrech spojení – obě strany si vymění informace
        - Typ souboru (textový vs. binární), vnitřní struktura souboru (obvykle bez struktury) a přenosový mód (proudový, blokový, komprimovaný)
        - Nezbytné pro překonání heterogenity komunikujících stran
- **Datová komunikace**
    - Proběhne podle požadavků, které na ten přenos byly domluveny v řídící komunikaci
- Je velmi důležitá standardizace těch protokolů, ta se děje pomocí RFC (request for comments)

11. přednáška

Mobilní systémy

Mobilita – schopnost přesunu malých zařízení, která máme s sebou jako uživatelé. Málokdy se chápe jako schopnost uživatele sednout si k různým fyzickým zařízením a pracovat na nich úplně stejně – tam, kde přestane na jednom, pokračuje na druhém. Zařízení nesmí být moc těžké, ale zároveň mít nějakou výdrž a výkon. Také musí vydržet něco bez připojení do elektrické sítě.

- **Inherentně distribuované**
    - Přiměřeně malá zařízení přenášená uživateli
    - Trvale nebo občas připojená do sítě
    - Omezená procesní kapacita
    - Využití bezdrátových protokolů
- **Klient/server nebo P2P** (peer to peer) model
    - Klient/server lépe přizpůsobitelný dostupnému výkonu mobilních zařízení – jedna výhoda – rozhodujeme, jaký výkon zajistí server a jaký klientské zařízení. U mobilních zařízení s nižším výkonem je tento model velmi lákavý – lze nastavit malá zátěž klientských zařízení. P2P výhoda – pokud jsou peery dostatečně blízko od sebe, stačí velmi nízký vyzařovací výkon antén, aby se dvě zařízení dokázala domluvit – malá spotřeba energie. Pokud by třeba komunikovaly přes P2P jen notebooky v posluchárně, bude to energeticky lepší, než přes přístupový bod.
    - Nutno ošetřit práci v odpojeném stavu – třeba v letadle, práce offline, stažení dokumentů na lokální disk
    - Vyvažování mezi kvalitou připojení a lokálně dostupným výkonem
- **Konvergence**
    - Růst výkonu malých zařízení (smartphones) – stírají se pomalu rozdíly mezi mobilním zařízením a desktopem. Je zde patrný velký vývoj – původně mobilita chápána jako že se pohybují lidé ke statickým zařízením, poté se chápala jako zařízení nesená lidmi, která se ale k síti připojují jenom někdy, a to v místě hotspotů (kde je síť dostupná), ale byla nutnost využívat zařízení i tam, kde síť není; až po dnešní stav, kdy může být zařízení online stále, a tudíž se mohou protokoly a komunikační záležitosti zjednodušit, protože se vlastně vrací do stejného stavu, ve kterém jsou imobilní

desktopy, které jsou připojeny k síti trvale. Nyní je problém s kapacitou, výkonem a velikostí, ne s připojením a protokoly.
    - Dostupnost a kvalita sítě – zajímá nás, k čemu je to zařízení připojeno. Naše zařízení se musí připojit na fixní bod, přes který může s jinými zařízeními komunikovat pomocí sítě (třeba wi-fi hotspot, jednotlivé stanice operátora – člověk ideálně ani nepozná přechod k jiné stanici).
    - Sdílení aplikací + synchronizace
        - Často s využitím nemobilního serveru

Senzorové sítě

- Malá (mobilní) zařízení sledující jisté parametry okolí – ta tvoří tu senzorovou síť
    - Teplota okolí (požáry) – v Austrálii používají senzory pro detekci lesních požárů. Vysypou se z letadla, jsou na nich čidla. Tyto fyzické senzory spolu začnou fyzicky komunikovat. Do lesa nemusíme nikdy fyzicky vkročit. Na kraj lesa dáme základnovou stanici. Pokud čidlo zpozoruje požár, přes ostatní senzory pošle informaci, že hoří, a svou GPS polohu. Otázka – jak napájet senzory a jak zabránit tomu, když někdo senzor najde a přiloží k němu zapalovač. Protokoly senzorových sítí počítají s tím, že drtivou většinu času senzory nic nevysílají, tak se také tolik nevybíjejí. Pokud senzor začne vysílat, teprve potom začne zjišťovat, jaké ostatní senzory jsou okolo něj, a ty, které nějakým způsobem ví, že jsou blíž základnové stanici než on, tak zprávu pošlou dále.
    - Tlak, vlhkost, … (budovy, stavby obecně, ale i sledování osob) – do malty, betonu, omítky se zalijí senzory (životnost 20-30 let), ty jsou citlivé na tlak – dělají kontrolu statiky. Často v oblastech s vysokou frekvencí zemětřesení, hurikánů; často bývá těžké určit, zda je budova výrazně poškozena, nebo ne.
- Komunikační zátěž
    - Omezená kapacita baterie
    - Nutné protokoly, které garantují „přiměřenou“ konektivitu, ale nezatěžují baterie
    - Cena výpočtu (vlastnost algoritmu/protokolu) vyjádřena ve Wattech, nikoliv počtu instrukcí
- Bezpečnostní aspekty
    - Senzoru se může útočník fyzicky zmocnit – s tím se musí počítat. Nasimulovat požár není snadné – více senzorů a jejich zapínání v určitém pořadí, v určitém místě
    - Kompromitace senzoru nesmí ohrozit celou síť

Kryptografie

Autentizace a autorizace

- Autentizace – prokázání, že „já“ jsem „já“ – to, co se běžně děje při přihlašování ke svému účtu na počítači zadáním hesla
- Autorizace – oprávnění přístupu ke službě/zdroji
- Delegace – prokázání, že já mohu vystupovat za někoho jiného

Kryptografie

- Ochrana komunikace
  - Snaha zajistit, že konkrétní zprávu si nemůže přečíst neoprávněná osoba
- Další požadavky na předávané zprávy:
  - **Integrita** – zpráva se nezmění, zjednodušeně řečeno
  - **Autenticita**
  - Non-repudiability – **nepopiratelnost** – pokud mi dojde zpráva, nemohu to popřít (jde to i druhým směrem)
- **Šifrování**
  - Zajišťuje pouze „nečitelnost“ zpráv

Symetrické a asymetrické šifry

- Šifrování pomocí sdíleného tajemství
  - Máme klíč a algoritmus, ten aplikujeme na zprávu
  - Stejný klíč pro šifrování a dešifrování
  - Je-li klíč delší než zpráva, nelze prolomit (velmi zjednodušeně) – přidání šumu
  - Problém distribuce (sdílení) klíče
- Asymetrická kryptografie
  - Máme dva klíče (soukromý a veřejný)
  - Soukromý má jen majitel klíče, veřejný je volně dostupný
  - Oba mohou být použity pro šifrování i dešifrování, ale komplementárně
    - Zpráva zašifrovaná soukromým klíčem je dešifrovatelná pouze veřejným klíčem a naopak
  - Problém, jak prokázat, komu patří konkrétní veřejný klíč

Symetrická kryptografie

- Aktuálně nejpoužívanější **AES** (**Rijndael**)
  - Starší např. DES, DES3
- Klíče délky 128-256 bitů (zpravidla)
- Rychlé algoritmy, snadno programovatelné přímo v HW
- Použití v autentizaci
  - Nepošlu přímo tajemství (heslo)
  - Jedna strana zvolí náhodné číslo, zašifruje a pošle
  - Druhá dešifruje, provede dohodnutou operaci, znovu zašifruje a pošle zpět
  - Příjemce dešifruje a zkontroluje výsledek
  - Popsaný proces je základem *Challenge-Response protokolu*
- Rizika/problémy
  - Distribuce hesla
  - Kompromitace hesla – kdo zná klíč, ví všechno, a nejde to rozeznat
  - Vícebodová komunikace

Asymetrická kryptografie

- Nemá jednoduchou analogii v reálném světě
- Používá jednosměrné funkce – funkce, u které lze spočítat funkční hodnotu, ale je mimořádně náročné spočítat její inverzní funkci. Např. vynásobení dvou hodně velkých prvočísel. Nejsou žádné jednoduché metody, které určí zpětně ta dvě prvočísla, trvá to velmi, velmi dlouho, třeba rozklad na prvočinitele.
- Klíče délky 2048-4096 bitů
- Složité algoritmy, náročná implementace
- Použití v autentizaci
    - Jedna strana zvolí náhodné číslo a zašifruje veřejným klíčem druhé strany
    - Druhá strana dešifruje svým soukromým, provede operaci a zašifruje veřejným klíčem první strany
    - První strana dešifruje svým soukromým klíčem a ověří
    - Pozor: popsaný princip je pouze jednostranná autentizace
- Rizika/problémy:
    - Autenticita veřejných klíčů
    - Nevhodné pro šifrování dlouhých zpráv

Digitální podpis

- Využití asymetrické kryptografie
- **Hash zprávy** – „otisk“ pevné délky
    - MD5, SHA1
    - Otisk je jedinečný pro konkrétní zprávu
    - Z otisku nelze rekonstruovat původní zprávu
- **Podpis:**
    - Ze zprávy proměnné délky vytvoříme „otisk“ pevné délky
    - Otisk zašifrujeme našim soukromým klíčem – podpis zprávy
- Ověření
    - Ze zprávy proměnné délky vytvoříme „otisk“ pevné délky
    - Vezmeme připojený podpis a dešifrujeme jej veřejným klíčem podpisujícího
    - Podpis je pravý, pokud se náš a dešifrovaný otisk shodují
- Princip použitelný i na garanci integrity a autenticity zprávy

Certifikační autorita

- Přiřazení veřejného klíče konkrétní entitě
- CA je institut, který
    - Ověří, kdo je vlastník soukromého klíče k určitému veřejnému
    - Vydá certifikát, tj. potvrzení o této vazbě, které sama podepíše
    - Např. pošle text, který zašifruji svým soukromým klíčem (ten ona nezná), ona ho pak dešifruje mým veřejným klíčem – pokud dostane nedostane požadovaný výsledek, tak neprojdu
- Jak věřit klíčům certifikačních autorit?
- Alternativy, např. **PGP (Pretty good privacy)**
    - Ring of trust

Delegace

- Potřebujeme pověřit nějakou entitu, aby mohla jednat našim jménem
- Naivní přístupy
    - Sdělíme sdílené tajemství
    - Svěříme soukromý klíč

  Nekorektní, nebezpečné a zpravidla jdou proti pravidlům
- Vydáme nový certifikát, který podepíšeme
    - Entita se prokazuje tímto novým (má jeho soukromý klíč)
    - Druhá strana vidí náš podpis pod delegací, proto akceptuje

Kombinace přístupů

- Jak zašifrujeme dlouhou zprávu?
    - Nejspíš symetrickým klíčem (rychlejší, méně výpočetně náročné)
- Jak ovšem ten klíč sdělíme druhé straně?
    - Nejlépe využitím asymetrické kryptografie
    - Veřejným klíčem druhé strany zašifrujeme symetrický klíč a přiložíme ke zprávě

Důvěryhodnost

- Proč máme primární a sekundární heslo do informačních systémů MU?
    - Častější použití klíče zvyšuje pravděpodobnost odchycení/zneužití
    - Některé systémy nemusí používat dostatečně spolehlivé systémy ověření (např. vyžadují poslání hesla)
- Souvisí s důvěryhodností
- Různé druhé strany považujeme za různě důvěryhodné
    - Snažíme se proto používat různé autentizační/komunikační mechanismy
    - Chráníme sdílená tajemství
- Exploze sdílených tajemství (loginů a hesel)
    - Vhodný kompromis pouze s několika úrovněmi
- Další přístupy
    - Digitální karty
    - Poskytovatelé identit, federace identit

12. přednáška

Multimediální systémy

- Cíl: přenos zvuku a obrazu po počítačové síti – postupně velká evoluce. Nyní třeba zaměření na to, jak přenášet haptická data (dotyk).
- Požadavky na kvalitu (vlastnosti) spojení
    - Včasné doručení – především u synchronních online přenosů videa a audia - videokonference
    - Nepříliš velký rozptyl doručení paketů – rozptyl zpoždění (jitter) – pokud si pouštíme hudbu bez bufferování (mezičlenu), tak někdy jdou pakety rychleji, jindy pomaleji, a sluch je dostatečně citlivý na to, aby ten rozdíl poznal.

- **Spojované sítě** (telefony) – před vlastním datovým přenosem dojde k fyzickému propojení celé trasy. Jestliže jsme předem schopni postavit trasu, tak známe její charakteristiku a víme, zda splníme dané požadavky nebo ne; případně se může stavět s ohledem na požadavky
    - Jednodušší řešení
    - Nedostatečná koncová kapacita
    - Potenciální plýtvání pásmem (musí být vyhrazeno, i když mlčíte)

Multimédia – podpora v IP sítích

- **Přepínané sítě** – žádné explicitní spojení, mezi dvěma uzly mohou pakety cestovat různými cestami, s různým zpožděním. Největší zpoždění dělá asi zpracování na jednotlivých směrovačích, ne fyzické cestování dat.
    - Mohou dobře využít *multicast* – dvou a vícecestná
    - Vyžadují kvalitu služby: *rezervace*
    - Možná řešení
        - **Overprovision** (dostatek kapacity bez ohledu na požadavky)
        - **Dedikované okruhy** (a la telefony): VPN
        - **Rezervace** pro každý tok zvlášť: RSVP – rezervační protokol
        - **Agregace** toků, rezervace (statická) pro agregace: DiffServ
    - Pro současný Internet vhodné poslední řešení, zbytek se špatně škáluje. Škálovatelná řešení jsou ta, která v sítích jako Internet vítězí.

Multimediální aplikace

- **Streaming**
    - Způsob doručení multimediálního obsahu klientům prostřednictvím sítě
    - *Live streaming*
        - Multimediální obsah vzniká živě během streamování
    - *Video on Demand* vs. *pasivní příjem*
        - Pasivní příjem obvykle pro příjem živých streamů
        - Možné streamovat i multimediální archivy
    - U streamingu je většinou jednostranný provoz. Je pro videokonferenci nepoužitelný, protože je zde poměrně velké zpoždění.
- **Videokonference**
    - Jednoznačný požadavek na interaktivou
    - Obousměrný provoz

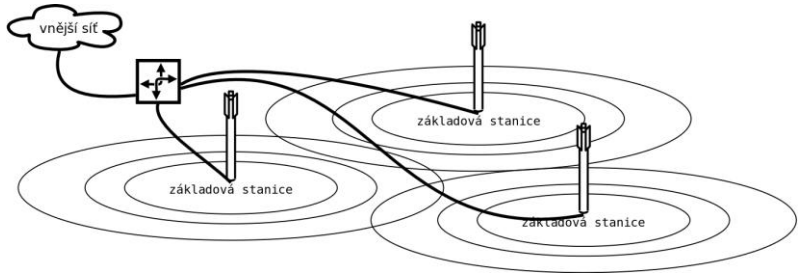
Bezdrátové sítě

- Cíl: umožnit přístup k výpočetním a komunikačním zdrojům z mobilních zařízení. Nyní už můžeme být online prakticky pořád, dříve jen někdy (někde).
    - **S infrastrukturou**: *buňková síť*
    - **Bez infrastruktury**: *ad-hoc sítě*
- Hlavní charakteristiky:
    - Podstatně vyšší chybovost než v drátových sítích
        - Oprava přímo na spojení, ne ve vyšších vrstvách
        - Často kombinováno s redundancí – data se vysílají v několika kopiích a některé protokoly jsou z toho schopny poskládat kompletní kopii.
    - Optické sítě (infra, laser)

        - Silně závislé na vnějších podmínkách (mlha)
    - Radiové
        - Kapacita závislá na frekvenci, kvalita na kódování a vyzářené energii

Buňková síť

- Základové stanice pokrývají území signálem
- Základny jsou propojené bezdrátovou sítí
- Veškerá komunikace mobilních agentů je směrována přes základové stanice
- Mobilní agent může plynule přecházet mezi základovými stanicemi
- Přijímač/vysílač je *single point of failure* – všechna komunikace s vnější sítí jde právě a pouze jen přes tento prvek. Pokud se pokazí, tak smůla.



Ad-hoc sítě

- Motivace: vytvořit síť při absenci infrastruktury
    - Živelné katastrofy, nedostatek financí/času – např. po hurikánu Katrina
    - Využívá pouze síťové vlastnosti účastníků – stačí jen správné protokoly a zařízení, není to nijak těžké
- Princip
    - Kolekce autonomních uzlů komunikujících skrze decentralizovanou **multi-hop síť**
    - Každý uzel zároveň koncovým uzlem i síťovým směrovačem
    - Dynamická topologie sítě
    - Řízení sítě rozděleno mezi jednotlivé uzly
- Výhody
    - Rychlé vybudování (a levné)
    - Odolnost – neobsahují single point of failure, kritická místa
    - Efektivní využívání rádiového spektra – není zde žádný centrální prvek, přes který by šla komunikace, každý se dorozumívá s těmi, které má v dosahu
- Nevýhody
    - Omezený dosah bezdrátové komunikace
    - Komplikované řízení sítě díky neexistenci centrální entity – uzly musí být schopny ohledat svoje okolí natolik, aby byly schopny směrovat určitým směrem pakety
    - Změny v topologii při pohybu mobilních uzlů
- Aplikace
    - Záchranné operace při přírodních katastrofách, protože tam byla ta infrastruktura zlikvidována
    - Zasíťování osobních zařízení (hodinky, PDA, medicínské přístroje, …)

- Vojenské operace – pokud armáda dobývá území, nemá na začátku čas budovat infrastrukturu a pokud nepřevezme stávající, musí vybudovat ad-hoc síť. Armáda tuto oblast posunula hodně dopředu.
- Senzorové sítě – senzory (třeba jen o velikosti mince) neslouží pro komunikaci s člověkem, ale s ostatními zařízeními; a další

Mobilní počítání

- Možné realizace:
  - Always-on – bezdrátové sítě
  - Přenos prostředí – realizované softwarově – přenáší se celé prostředí, není potřeba být stále online
- Mobilita s přenosem prostředí
  - Např. čtení pošty přes webový prohlížeč – v době načtení/odesílání je potřeba připojení, jindy ne – proces čtení není kontinuální
  - Problémy
    - Různost klientských systémů
    - Bezpečnost – autentizace uživatele; obecně nejsou bezdrátové sítě příliš bezpečné a jsou vcelku snadno odposlechnutelné
    - Vnímaná kvalita závislá na kvalitě připojení

Distribuované systémy

- Počítač: několik vzájemně propojených komponent
- Co se stane, když některé z propojení nahradíme sítí? Vznikne distribuovaný systém
- Definice:
  - Systém, který je tvořen dvěma nebo více nezávislými počítači propojenými sítí a komunikujícími formou předávání zpráv
  - Distribuovaný systém tvoří nezávislé počítače, které se uživateli jeví jako jeden celek (Tanenbaum). Jak to? Je to dáno přístupem. Jestliže máme SW, který nám udělá z těch počítačů jeden celek a vytvoří jeden virtuální počítač, potom se to uživateli jeví jako jeden počítač.

Příklady

- **Internet** – nevnímáme jeho vnitřní strukturu (uzly), vidíme jen ty dva koncové uzly
- Telefonní systém (automatické ústředny)
- Multimediální systémy (videokonference, e-learning)
- Mobilní systémy
- Clustery
- Gridy – výpočetní mřížky. Dříve představa, že tak, jak máme elektrickou rozvodnou síť, tak budeme mít počítače propojené internetem, které přivedou výpočetní kapacity všude, kde to bude potřeba. Dnes už je to překonaná představa, přesto gridy jako takové se stále používají.
- Peer-to-peer systémy – virtuální ad-hoc síť; na aplikační úrovni spojuje dva uzly, které spolu přímo komunikují za účelem sdílení datového obsahu, sdílení výpočetních a přístupových kapacit, atd.
- Cloud

Problémy distribuovaných systémů

- **Heterogenita** jednotlivých složek – platí např. u mobilních sítí, každá z entit má k dispozici jinou výpočetní kapacitu a tyto různé entity spolu musí spolupracovat
    - Middleware: skrývá heterogenitu (CORBA, Globus) – řeší problém heterogenity. Na horní úrovni dává službu jednoho celku
    - Mobilní kód (Java)
- **Otevřenost**/interoperabilita
    - Nezbytné využití standardů
- **Bezpečnost**
    - Autentizace, autorizace, soukromí – jakmile máme více složek, je větší šance, že některé z nich nepracují tak, jak mají. Je daleko jednodušší zajistit bezpečnost jednoho uzlu, než padesáti tisíc uzlů najednou.
- **Zpracování výpadků**
    - Detekce, maskování, tolerance; jeden výpadek nesmí ohrozit činnost celého systému
- **Rozšiřitelnost**
- **Paralelismus** – např. člověk přemýšlí sériově, tak se nám také lépe programuje sériově
    - Nebezpečí např. „smrtelného objetí“ (*deadlock*) – to pokud se to naprogramuje špatně. Dva systémy vzájemně čekají na prostředky toho druhého systému.
    - Závislosti (synchronní pohled) – abychom nevnesli do výpočtů chyby
- **Transparence**
    - Přístup
    - Místo
    - Replikace
    - Selhání
    - Mobilita/přenositelnost
    - Výkon
    - Škálovatelnost/rozšiřitelnost

Gridy

- Motivace: sdílení výpočetních zdrojů za účelem zvýšení efektivity
    - Inspirace z elektrické rozvodné sítě (*power grid*)
- Vlastnosti
    - Rozsáhlé distribuované systémy
        - Heterogenní – připojeny jsou různé systémy
        - Geograficky rozsáhlé
        - Dynamické (z pohledu uživatele)
    - Velký výkon (desítky tisíc procesorů) – využívá jej např. CERN
    - Velké datové objemy (petabyty a více)
- Příklady:
    - **Data Gridy** – zpracování velkých objemů dat, generovaných
        - Zařízeními částicové fyziky - CERN
        - Radioteleskopy
        - Analýzou genomu
        - 3D snímky (mozek)

- **Výpočetní Gridy** – menší objem dat, ale náročné výpočty
  - Astronomie (částečně i Data Gridy)
  - Vlastnosti materiálů
  - Předpověď počasí (též Data Grid)
  - Struktura a chování molekul – výpočetní chemie

Cloud computing

- Nový přístup k nabídce výpočetních a úložných služeb – dříve byl problém odlišit cloud od gridu. Liší se především *virtualizací*. Na jednotlivých uzlech jsou virtualizovány zdroje (počítače) tak, aby byla zachována spravovatelnost.
- Postaven na virtualizaci zdrojů
  - Ta umožňuje nabídnout počítač nebo celou skupinu počítačů (*cluster, grid*) při zachování spravovatelnosti (*manageability*)
  - Uživatel dostává „holý“ systém, který sám spravuje
  - Jednoduchý přístup, zpravidla přes webové rozhraní
  - *Pay per use*, tj. žádné počáteční investiční náklady
- Příklad
  - **Amazon Elastic Cloud**
    - Přístup přes webové rozhraní
    - Platba kreditní kartou
    - Součástí nabídky i úložný prostor (Amazon S3)
      - Nestrukturované objekty (upload/download), blokový systém (holý disk), filesystém
      - Platba i za přenos dat z/do S3, nikoliv za interní přesuny dat
- Cloudy nabízí flexibilní kapacity
  - Je možno okamžitě dokoupit další zdroje
  - Virtualizace podporuje navyšování výkonu poskytnutím kopií
  - Potenciál pro odolnost proti výpadku
- Otevřené bezpečnostní problémy
  - Nejistota, kde jsou skutečně data uložena
  - Data i výpočty de-facto outsourcovány – ztráta kontroly
- Vhodné zejména tam, kde bezpečnost není kritická a není možné předem odhadnout skutečnou potřebu zejména výpočetního výkonu (a ta silně kolísá v čase)

Peer-to-peer (P2P) systémy

- Decentralizovaný distribuovaný systém: klient-klient
- Tvořen vzájemně komunikujícími identickými entitami (peery)
- Opak modelu klient-server
- Každý peer je zároveň serverem i klientem
  - Poskytuje služby ostatním peerům – role serveru
  - Využívá služby ostatních peerů – role klienta
  - Existují i hybridní systémy, které využívají kousek od obou systémů. *Superuzly* (ultrauzly) – mají více „síly“ než normální uzly a umožní klientům přímou komunikaci mezi sebou. Např. *Gnutela*.

- Příklady
  - **Skype** – přenos hlasu a obrazu v reálném čase
  - BOINC – platforma pro distribuované výpočty
  - BitTorrent – sdílení dat
  - Bitcoin – digitální měna
  - …
- **Vlastnosti P2P systémů:**
  - Distribuované řízení – neexistence centrální entity
  - Samoorganizace
  - Heterogenita – peerové běží na různých platformách
  - Škálovatelnost – nehrozí přetížení centrální entity
  - Dynamika – topologie systému se velmi rychle mění
  - Sdílení zdrojů – každý peer se svými zdroji podílí na fungování P2P systému

Klient-server vs. Peer-to-peer

- Náročnost zbudování
  - K-S využívá jednoduchých modelů komunikace
  - P2P vyžaduje komplexní interakce
- Spravovatelnost
  - Správa K-S systému je přehlednější díky koncentraci komunikace v jednom bodě
- Škálovatelnost
  - K-S model limitován HW parametry serveru – využívá se vyvažování zátěže mezi několika fyzickými stroji. Nikdy v tomto systému nedochází ke komunikaci mezi klienty rovnou. Pokud je klientů příliš, servery si rozdělují požadavky, aby nedošlo k přehlcení.
  - P2P systém škáluje z principu – s rostoucím počtem peerů roste kapacita systému. Nemá žádný centrální prvek = to, co by mohlo být přetíženo. Každý uzel se podílí na fungování sítě.
- Bezpečnost
  - V K-S modelu je za bezpečnost zodpovědný server
  - V P2P systému je zodpovědnost rozložena mezi peery – nutnost komplexnějších bezpečnostních protokolů
- Spolehlivost
  - K-S systém je závislý na běhu serveru – single point of failure
  - P2P systém je do velké míry redundantní – jednu funkcionalitu poskytuje zároveň více peerů

13. přednáška

IT a společnost

- Celospolečenský dopad
- Bezprostřední vliv na:
  - Komunikace
  - Zpracování informací
  - Výuku a vzdělávání
  - Výzkum
  - Ekonomické procesy
- Obdobně jako nástup průmyslu vnucuje celé společnosti změnu
- Rozdíl v rychlosti změny a příhraničním přesahu

Informační společnost

- Informace se stávají nejcennější „surovinou“
  - Informace zvyšuje nebo vytváří hodnotu
- Roste význam
  - Sběru informací
  - Zpracování informací – znalosti
  - Distribuce informací
- Roste i počet subjektů, které mají informace k dispozici
- Šíření informací:
  - Informace se šíří rychleji než znalosti
  - Cena za šíření je minimální a nesouvisí s cenou informace
  - Rozhodování se na základě známých informací není vždy totéž jako rozhodování se na základě znalostí
  - Příklad: *Volatilita burzy*
- Cenzura a autocenzura
  - Snaha odlišit „škodlivé“ informace
    - Subjektivní pojem, souvisí s kulturou a lokální společností
  - „Žumpy“ diskuzních fór

Nová ekonomika

- Zpracování informací (a následné rozhodování)
- Informace místo fyzických objektů
  - Podstatně rychlejší
  - Podstatně levnější
  - Užší propojení jednotlivých částí ekonomického řetězce
  - Příklady
    - Bezpapírová kancelář (vyžaduje změnu přístupu)
    - Virtuální a rozšířená (augmented) realita
  - Postupné „vstřebávání IT“ pomalejší, než se očekávalo
- Masové IT systémy jako další zdroj bohatství

Komunikace a síťové prostředky

- Sociální důsledky
  - **Sblížení** – telefon versus klasická pošta
  - **Zrychlení** – přeprava dokumentů faxem nebo klasickou poštou
  - **Mobilita** – dostupnost vždy a všude (mobilní telefony)
  - Vyvrcholení **v počítačem podporované komunikaci**
  - **Překonání handicapů** (fyzických, místních, jazykových, …)
  - Paradoxně i **depersonalizace** – nevíte, s kým tak rychle komunikujete
- **Sociální sítě**
  - Pocit blízkosti
  - Ochota sdílet soukromé informace
  - Morální a bezpečnostní důsledky
- Teprve se učíme reagovat na změnu

„Nová“ věda

- Simulace experimentů
  - Chápáno jako fundamentálně nová vědecká metoda
- Realizace „nerealizovatelného“
  - Příliš nebezpečné děje (výbuch supernovy)
  - Příliš rychlé/pomalé děje
  - Komplexní simulace
- (Mezinárodní) spolupráce – možnost tvorby rozsáhlých týmů
  - e-Infrastruktura jako páteř (síť jen začátek)
  - Sdílení a přístup k jedinečným přístrojům a systémům
- Pojem e-science

Profesní organizace

- **ACM** (Association for Computing Machinery)
- **IEEE CS** (IEEE Computing Society)
- **IFIP, ERCIM**, …
- V ČR např. Společnost pro kybernetiku, Informatická společnost

ACM (Association of Computing Machinery)

„ACM, the world’s largest educational and scientific computing society, delivers resources that advance computing as a science and a profession. ACM provides the computing field's premier Digital Library and serves its members and the computing profession with leading-edge publications, conferences, and career resources.“

- Založena 1947
- Cca 70 000 členů
- http://www.acm.org

IEEE (Institute of Electrical and Eletronics Engineers)

- Cca 370 000 členů
- http://www.ieee.org
- IEEE CS (Computing Society)
    - Založena 1946
    - Cca 100 000 členů (60% v USA)
    - http://www.computer.org

Etické kódy

- General ACM Code of Ethics and Professional Conduct
    - Schválen 16. října 1992
- Software Engineering Code of Ethics and Professional Practice
    - Verze 5.2
    - Společně s ACM a IEEE CS

Etický kód ACM

- Obecné morální imperativy
- Profesní zodpovědnost
- Odpovědnost vedoucích

- **Obecná část:**
    - Contribute to society and human well-being
    - Avoid harm to others
    - Be honest and trustworthy
    - Be fair and take action not to discriminate
    - Honor property rights including copyrights and patent
    - Give proper credit for intellectual property
    - Respect the privacy of others
    - Honor confidentiality

- **Profesní část:**
    - Strive to achieve the highest quality, effectiveness and dignity in both the process and products of professional work
    - Acquire and maintain professional competence
    - Know and respect existing laws pertaining to professional work
    - Accept and provide appropriate professional review
    - Give comprehensive and thorough evaluations of computer systems and their impacts, including analysis of possible risks
    - Honor contracts, agreements, and assigned responsibilities
    - Improve public understanding of computing and its consequences
    - Access computing and communication resources only when authorized to do so

- **Vedoucí:**
  - Articulate social responsibilities of members of an organizational unit and encourage full acceptance of those responsibilities
  - Manage personnel and resources to design and build information systems that enhance the quality of working life.
  - Acknowledge and support proper and authorized uses of an organization's computing and communication resources
  - Ensure that users and those who will be affected by a system have their needs clearly articulated during the assessment and design of requirements; later the system must be validated to meet requirements
  - Articulate and support policies that protect the dignity of users and others affected by a computing system
  - Create opportunities for members of the organization to learn the principles and limitations of computer systems

- **Etický kód programátorů:**
  - 1. PUBLIC { Software engineers shall act consistently with the public interest.
  - 2. CLIENT AND EMPLOYER { Software engineers shall act in a manner that is in the best interests of their client and employer consistent with the public interest.
  - 3. PRODUCT { Software engineers shall ensure that their products and related modifications meet the highest professional standards possible.
  - 4. JUDGMENT { Software engineers shall maintain integrity and independence in their professional judgment.
  - 5. MANAGEMENT { Software engineering managers and leaders shall subscribe to and promote an ethical approach to the management of software development and maintenance.
  - 6. PROFESSION { Software engineers shall advance the integrity and reputation of the profession consistent with the public interest.
  - 7. COLLEAGUES { Software engineers shall be fair to and supportive of their colleagues.
  - 8. SELF { Software engineers shall participate in lifelong leasing regarding the practice of their profession and shall promote an ethical approach to the practice of the profession.