OS – Design a bezpečnost

Tomáš Hudec

Tomas.Hudec@upce.cz

http://fei-as.upceucebny.cz/usr/hudec/vyuka/os/

Podstata problému designování

- je důležité mít jasný cíl, co přesně požadujeme
  - úspěch jazyka C (Ken Thompson, Dennis Ritchie)
    - jasný cíl: systémové programování
    - vyšší programovací jazyk pro napsání OS (UNIX)
  - neúspěch jazyka PL/I (IBM 1960)
    - potřeba podpory jazyků Fortran a Cobol
    - někteří tvrdili, že Algol je lepší než oba dva
    - IBM sestavila komisi pro navrhnutí jazyka pro všechny:
    - vzít to nejlepší z jazyků Fortran, Cobol a Algol
    - žádná sjednocující myšlenka (vize)

Cíle při návrhu OS

- cíle se liší v závislosti na určení OS
  - vestavěné, serverové, osobní systémy
- hlavní společné cíle při navrhování OS
  - definovat abstrakce
  - poskytnout základní (primitivní) operace
  - zajistit izolaci
  - spravovat hardware

Abstrakce s operacemi

- definování abstrakcí
  - zřejmě nejnáročnější část
    - definování procesů, souborů – poměrně zřejmé
    - další abstrakce: vlákna, synchronizace, signály, paměťový model, V/V, IPC, …
  - abstrakce jsou obvykle datové struktury
- poskytnutí základních (primitivních) operací
  - jednoduché operace nad strukturami (abstrakcemi)
  - operace jsou uživateli (programátorovi) přístupné skrz systémová volání

Izolace

- zajištění izolace
  - uživatelé systému si nesmí zasahovat do svých procesů, systémových prostředků (zdrojů) a dat
    - chyba procesu uživatele neovlivní zbytek systému
    - data se obvykle ukládají do souborů
    - alokované zdroje eviduje struktura procesu
  - izolují se tedy procesy (jejich alokované zdroje)
  - poskytuje se řízení ochrany (přístupová práva)
    - dat, souborů, ale i dalších prostředků a operací nad nimi
  - nabídnout možnost sdílení
    - izolace tedy musí být selektivní

Správa HW

- spravování HW
  - čipy pro řadiče přerušení, systémové sběrnice, vstupně-výstupní zařízení
  - ovladače zařízení (device drivers)
  - abstrakce HW

Proč je těžké navrhnout OS?

- pro vývoj HW – Moorův zákon
- nikdo nedefinoval zákon o zlepšování OS
  - i v současnosti existují OS, které jsou v klíčových ohledech (jako spolehlivost) horší než UNIX Version 7 ze 70. let 20. století
  - Proč?
- důvodů je celá řada
  - včetně nedodržení dobrých designových principů
  - OS je od podstaty jiný než aplikace
    - rozsáhlost, složitost, …

Důvody náročnosti návrhu OS

1. rozsáhlost, komplexnost, složitost
2. konkurence – uživatelé, procesy, prostředky
3. ochrana – nepřátelské prostředí
4. sdílení dat a prostředků
5. životnost
6. obecnost použití
7. přenositelnost
8. zpětná kompatibilita

Principy a vodítka (1)

- jednoduchost
  - Antoine de St. Exupéry: „*Dokonalosti nedosáhneme tehdy, když není nic, co bychom přidali, ale tehdy, když už není, co bychom odebrali.*“
  - méně je více – KISS (Keep It Simple, Stupid)
- úplnost
  - Albert Einstein: „*Vše by mělo být tak jednoduché, jak to jde, ale ne jednodušší.*“
  - OS by měl dělat to, co má, ale nic navíc
    - minimum mechanismů s jasným chováním

Principy a vodítka (2)

- jednoduchost a úplnost
  - každá část, vlastnost, funkce, systémové volání by měla dělat jednu věc a tu dělat dobře a správně
  - přidáváme-li novou vlastnost, je třeba položit si otázku: „Co hrozného by se stalo, když to nepřidáme?“
  - je-li odpověď:
    „Nic, ale někdo by to někdy mohl potřebovat.“,
    pak je lepší tohle zahrnout do knihovny,
    nikoliv do jádra OS (i když to bude pomalejší)

Principy a vodítka (3)

- jednoduchost × úplnost
  - Tony Hoare: „*There are two ways of constructing a software design:*
    - *One way is to make it so simple that there are obviously no deficiencies, and*
    - *the other way is to make it so complicated that there are no obvious deficiencies.*
    - *The first method is far more difficult.“*

Principy a vodítka – příklady

- MINIX
  - systém je sada procesů se správou paměti, souborovým systémem a ovladači, které běží jako samostatné procesy
  - tři systémová volání (send, receive, sendrec)
    - systém řeší předávání zpráv mezi subsystémy
    - stačí dvě – sendrec je pouhá optimalizace, aby stačil jeden TRAP (skok do jádra)
- Amoeba
  - jediné systémové volání – RPC (Remote Procedure Call) – velmi podobné sendrec ze systému MINIX

Principy a vodítka (4)

- efektivita implementace
  - pokud by měla být vlastnost (systémové volání) neefektivní, pravděpodobně nemá cenu ji mít
  - programátor (uživatel OS) by měl také vědět cenu těchto volání
    - např. posunutí pozice v souboru:
    - lseek – prosté posunutí pozice (změna proměnné)
    - read – provedení V/V operace
  - je-li intuitivní předpoklad ceny volání špatný, budou psát programátoři neefektivní programy

Paradigmata – sladění systému

- architectural coherence
  - architektonická souvislost (promyšlenost)
    - jak jsou vlastnosti systému sladěny v celek
  - pohled zákazníka na systém, zákazníci jsou
    - uživatelé (spouštějí aplikace) – sledují GUI
    - programátoři (píší aplikace) – používají systémová volání
- paradigma uživatelského rozhraní
  - top-down design (od GUI k systémovým voláním)
  - bottom-up design (od systémových volání ke GUI)

Paradigmata (různá pojetí)

- spouštěcí paradigma – execution paradigm
  - algoritmický přístup
    - provádí se algoritmus, volají se podprogramy, …
    - OS je pro programátora serverem, poskytuje služby ve formě systémových volání
  - řízení událostmi
    - v cyklu se zpracovávají události a reaguje se na ně
    - vhodné pro interaktivní systémy
- datové paradigma
  - jak přistupovat k datům a periferiím
    - např. UNIX: „vše je soubor“

Bezpečnost

- cíle:
  - důvěrnost dat
  - integrita dat
  - dostupnost systému
- příčiny ztráty dat:
  - vyšší moc – živelní pohroma, válka, …
  - HW či SW chyby – CPU, disky, chyby SW, …
  - lidský faktor – chybné vstupy, ztráta disku, počítače, …

Útoky na systém (1)

- útoky zvenčí
- útoky zevnitř
  - od uživatelů, pro které je systém určen!
- způsoby napadení
  - trojské koně
    - nevinně vypadající programy, které mají „funkci“ navíc
  - login spoofing
    - imitace přihlašovací obrazovky OS
  - viry, červi

Útoky na systém (2)

- způsoby napadení
  - zadní vrátka (trap door, back door)
    - vložená metoda přístupu navíc, např. při speciální kombinaci vstupních dat se zaručí další práva
  - logické (časované) bomby
    - po určité době nebo za určitých okolností se spouští škodlivý kód
  - buffer / stack overflow
    - chyba neošetření maximální délky vstupu
    - delší vstup pak přepíše další proměnné nebo dokonce kód a způsobí neočekávané chování programu

Útoky na systém (3)

- způsoby napadení
  - skenování portů (port scanning)
    - zjišťování spuštěných služeb a jejich obslužných programů pro následný pokus o využití známé bezpečnostní slabiny
  - DoS / DDoS – (Distributed) Denial of Service
    - zahlcení serveru množstvím požadavků, takže se služba zastaví nebo alespoň velmi zpomalí
    - distribuovaná verze – požadavky přicházejí z mnoha různých (již napadených) uzlů sítě
    - při podvržení adresy lze dosáhnout blokace části sítě

Metody útoku

- pasivní odposlech dat (neautorizovaný)
- aktivní odposlech
  - změna zprávy nebo její části
  - podvržení zprávy (s platnými adresami)
  - znovuzaslání dříve zachycené zprávy
  - vydávání se za jiný uzel dané sítě

Proslulé útoky na UNIX

- lpr a /etc/passwd
  - tiskový démon umožňoval tisk všem uživatelům a také odstranění souboru
- core a /etc/passwd
  - po pádu procesu systém uloží soubor core (obraz paměti procesu)
- mkdir a pomalý systém
  - vytváření nového i-uzlu jako root

Techniky ochrany

- domény ochrany
  - stanovení domén a práv
- ACL – Access Control List
  - stanovení seznamu práv na objekty pro uživatele
- capability – C-list
  - stanovení práv procesům
    - MAC (Mandatory Access Control)
    - Linux od r. 1998: SELinux (NSA), AppArmor (SuSE)
- firewally a komunikační filtry

Ochrana uživatele

- autentizace (autentifikace) – ověření identity
  - jméno a heslo – důležité je užívat správná hesla
  - hesla na jedno použití (tokens)
  - soukromý klíč (např. SSH)
  - použití smart card nebo podobného HW
  - biometrie – otisky prstů, oční sítnice apod.
- autorizace – ověření oprávnění

Ochrana dat

- zamezení přístupu (přístupová práva)
  - lze jen v systému
  - je-li systém zcizen nebo se data přenášejí, je třeba použít jinou ochranu
- šifrování
  - symetrická kryptografie
  - asymetrická kryptografie
  - jednosměrné funkce
  - digitální podpisy

Šifrování

- kódování × šifrování
- označení:
  - M = zpráva
  - Ke = šifrovací klíč, Kd = dešifrovací klíč
  - E = šifrovací funkce, D = dešifrovací funkce
  - S = zašifrovaná zpráva, $S = E(M, Ke)$
  - dešifrování: $D(S, Kd) = D(E(M, Ke), Kd) = M$
  - symetrická kryptografie: tajný klíč $Ke = Kd$
  - asymetrická krypt.: veřejný klíč $Ke \neq$ tajný klíč Kd

Šifrování – příklad

- princip zaslání zašifrované zprávy od A k B
  - příjemce B si vygeneruje klíče Ke_B a Kd_B
  - klíč Ke_B příjemce zveřejní, Kd_B si ponechá v tajnosti
  - odesílatel A získá veřejný klíč Ke_B příjemce B
  - A zašle tajnou zprávu $S = E(M, Ke_B)$
  - zpráva lze dešifrovat pouze klíčem Kd_B, takže ji získá pouze příjemce B: $D(S, Kd_B) = M$

Podepisování – příklad

- princip zaslání podepsané zprávy od A k B
  - odesílatel A si vygeneruje klíče Ke_A a Kd_A
  - klíč Ke_A odesílatel zveřejní, Kd_A si ponechá v tajnosti
  - A zašle (veřejnou) zprávu M a připojí k ní podpis $S = D(h(M), Kd_A)$, kde h je vhodná hashovací funkce
  - příjemce B získá veřejný klíč Ke_A odesílatele A
  - B ověří původ zprávy tak, že získá $h' = E(S, Ke_A)$ a následně porovná $h(M)$ s h'
  - útočník nezná Kd_A, tudíž nemůže zfalšovat podpis

Šifrování – podmínky

- pokud máme *M* i *S*, pak získat klíče *Ke* a *Kd* je výpočetně náročné
- generování párů klíčů *Ke* a *Kd* musí být snadné
  - $D(E(M, Ke), Kd) = M$
  - klíče *Ke* a *Kd* nesmí být od sebe odvoditelné
- algoritmy *E* a *D* musí být efektivní, snadno použitelné a inverzní – $E(D(M, Kd), Ke) = D(E(M, Ke), Kd)$
- máme-li klíč *Ke*, musí být problém určení odpovídajícího klíče *Kd* výpočetně náročný

Šifrování – výpočetní náročnost

- Co znamená výpočetně náročný problém?
  - řešení nelze snadno spočítat v konečném čase
- příklady
  - problém faktorizace součinu obrovských prvočísel
  - knapsack problem (problém batohu se závažími)
  - další NP-úplné problémy (NP = nedeterministicky polynomiální)
- NP-úplnost – NP je třída obtížnosti
  - úplné nedeterministicky polynomiální problémy jsou **zjednodušeně řečeno** ty nejtěžší problémy

Třídy složitosti

- třída P
  - všechny úlohy lze řešit v polynomiálně omezeném čase na **deterministickém** Turingově stroji
- třída NP
  - úlohy lze řešit v polynomiálně omezeném čase na **nedeterministickém** Turingově stroji
  - jsou to ty problémy, jejichž **řešení lze ověřit v polynomiálním čase**, ovšem nevíme, zda je lze také v polynomiálním čase vyřešit

Turingův stroj

- teoretický model počítače popsaný matematikem Alanem Turingem
  - skládá se z konečného automatu, programu ve tvaru pravidel přechodové funkce a pravostranně nekonečné pásky pro zápis mezivýsledků
- nedeterministický Turingův stroj
  - umožňuje v každém kroku rozvětvit výpočet na *n* větví, v nichž se posléze řešení hledá současně
  - ekvivalentně se hovoří o stroji, který na místě rozhodování uhodne správnou cestu výpočtu

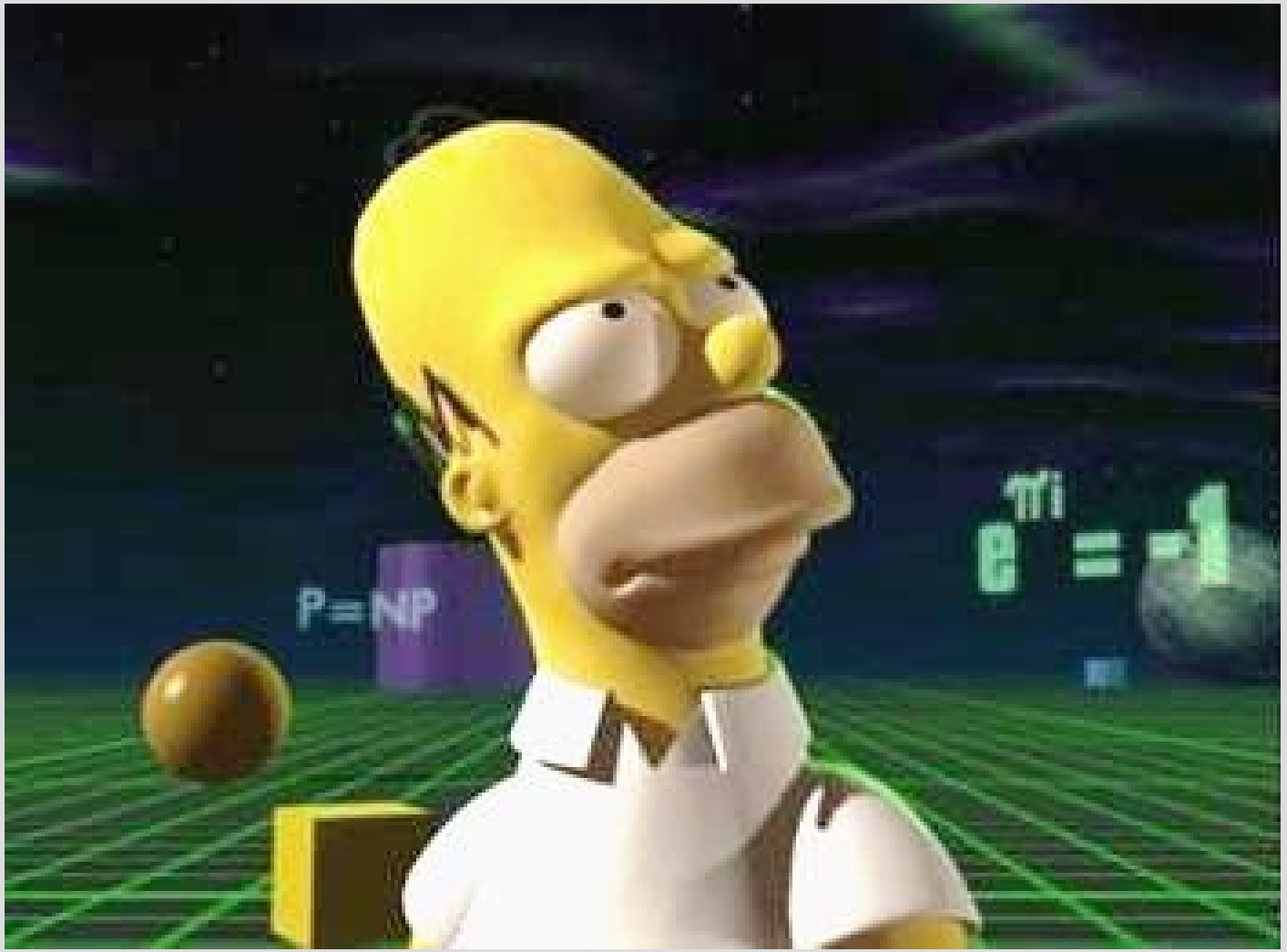
NP-úplné problémy

- NP-úplné (NP-complete, NPC) problémy jsou
  - takové nedeterministicky polynomiální problémy, na které jsou polynomiálně redukovatelné všechny ostatní problémy z třídy NP
  - třídu NP-úplných úloh tvoří v jistém smyslu ty nejtěžší úlohy z NP
  - pro svou složitost (obtížnost nalezení řešení) se využívají v kryptografii

Problém P versus NP

- v teoretické informatice se takto označuje otázka, zda platí rovnost P = NP
- považuje se za nejdůležitější otevřený problém tohoto oboru
  - zařazený mezi sedm tzv. problémů tisíciletí
  - Clay Mathematics Institute vypsal 24. května 2000 za rozhodnutí vztahu odměnu 1 000 000 dolarů
  - je zřejmé, že $P \subseteq NP$
  - předpokládá se, že $NP \neq P$

Problém P versus NP (obrázek)



Problém P versus NP

- nalezení **deterministického** polynomiálního algoritmu pro libovolnou NP-úplnou úlohu znamená
  - všechny nedeterministicky polynomiální problémy jsou řešitelné v polynomiálním čase
  - tedy třída NP se „zhroutí“ do třídy P: P = NP
  - důsledek: zhroucení dnešní kryptografie

Šifrování – příklad s faktorizací

- zvolíme prvočísla p, q — $p = 3$, $q = 11$
- vypočítáme $n = p \cdot q$ — $n = 3 \cdot 11 = 33$
- vybereme libovolné Kd nesoudělné s $L(n)$: — Kd = např. 13
  - $L(n) = (p - 1)(q - 1)$ — NSD(13, 20) = 1
  - $\max(p, q) < Kd < L(n)$ — $L(n) = 2 \cdot 10 = 20$; $11 < Kd < 20$
- vypočítáme Ke: — $Ke = 17$
  - $0 < Ke < L(n)$ — $0 < Ke < 20$
  - $Ke \cdot Kd = 1 \pmod{L(n)}$ — $17 \cdot 13 = 1 \pmod{20}$

Šifrování – příklad s faktorizací

- nyní můžeme šifrovat hodnoty 0 až (n − 1)
  - šifrování znaku zprávy M_i: $S_i = M_i^{Ke} \pmod{n}$
  - dešifrování znaku zprávy S_i: $M_i = S_i^{Kd} \pmod{n}$
- *Ke* a *n* jsou známé
- *Kd* lze těžko odvodit, protože neznáme *p* a *q*
- např. šifrujeme „ZDAR“ (A = 1, B = 2, …)
  - *E*("26 04 01 18", 17, 33) = "05 16 01 06" = „EPAD“
  - *D*("05 16 01 06", 13, 33) = "26 04 01 18" = „ZDAR“