

PB001: Úvod do informačních technologií

Luděk Matyska (Eva Hladká)

Fakulta informatiky Masarykovy univerzity

podzim 2016



EVROPSKÁ UNIE



MINISTERSTVO ŠKOLSTVÍ,
MLÁDEŽE A TĚLOVÝCHOVY



OP Vzdělávání
pro konkurenceschopnost



INVESTICE DO ROZVOJE VZDĚLÁVÁNÍ

Obsah přednášky

- 1 Klasifikace OS
- 2 Kernel operačního systému
- 3 Procesy
- 4 Správa paměti
- 5 Systém souborů
- 6 Přerušování
- 7 Problém časování
- 8 Programové vybavení

Klasifikace OS

- Monolitický
- Vrstvený
- Modulární
- Koncept kernelu a mikro-kernelu

Monolický OS

- Původní operační systémy (proprietární)
- Abstrakce nepoužívána příliš *dovnitř*
 - jedna skupina “opravdových programátorů” po celou dobu životnosti OS
- Nejasné rozlišení funkcí uvnitř operačního systému
- „Velké“, špatně rozšiřitelné, špatně udržovatelné
- Poplatné době pomalejšího vývoje hardware a jeho vysoké ceny

Vrstvený OS

- Vrstvy odpovídají procesům správy:
 - Správa CPU
 - Správa paměti
 - Správa periférií
 - Správa systému souborů
- Lepší abstrakce
- Komunikace mezi vrstvami
 - Komplikuje strukturu
 - Riziko obcházení (shortcuts)
 - Jistá penalizace ve výkonu

Modulární OS

- Moduly namísto vrstev
- Zapouzdření (enkapsulace) funkcí
- Komunikace mezi moduly
 - Složitější na obejití
 - Může mít vyšší režii (vyšší penalizace ve výkonu)
- Příbuzný objektovému přístupu
- Lepší údržba
 - Moduly menší, snáze se vyměňují než celé vrstvy
- Riziko vzniku „fatware“
 - Příliš mnoho příliš malých modulů

Kernel operačního systému

- Kernel, též *jádro* operačního systému:
 - Základní složka operačního systému
 - Odpovídá za:
 - Alokaci a správu zdrojů
 - Přímé ovládání hardware (nízkoúrovňové interfaces)
 - Bezpečnost
- Mikrokernel:
 - *Malé je pěkné*
 - 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 operačnímu systému 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)

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 nbytes)`
 - 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
- Příznaky a klíče pro ovládání specifických vlastností: přenositelnost versus efektivita
- Ovladače na nejnižší úrovni („nejblíže“ hardware)
 - Specifické „jazyky“ ovládání periferií na této úrovni
 - Práce se *signály* (např. změna stavu periferie)

Periferie z pohledu (modulárního) OS

Začlenění ovladače do jádra

- kooperativní vs. hierarchické (možnost preempce)
- 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

OS: Procesy

- Proces je abstrakce průchodu programem
 - 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 paměť pro data

Typy procesů

- Klasické (heavy-weight) procesy (např. UNIX)
 - Všechna data privátní
 - Sdílen pouze program (read-only)
- *Lehké* (light-weight) procesy či Vlákna (threads)
 - Minimum vlastní paměti
 - Většina dat sdílena

Procesy detailněji

- Vytvoření procesu
 - `fork()` a jeho varianty
 - *Přesná* kopie původního procesu
 - *Rodič* a *potomek*
 - První proces v OS vytvářen jinak (`init` v Unixu)
- Stav
 - Start/vytvoření, připraven (`ready`), běží (`running`), je blokován (čeká), skončil

Synchronizace – problém

- Race condition: soupeření v čase
 - Proces P {
 Load RegistrA, X
 Load RegistrB, Y
 Add RegistrA, RegistrB
 Store RegistrA, X # X+=Y
}
- Dvě instance procesu P, používají stejná X a Y
- Nedefinovatelné výsledky
 - Je-li na začátku $X=Y=1$, pak na konci může být $X=2$ nebo $X=3$

Synchronizace – řešení

- Kritická sekce
 - Semaforey: celočíselné proměnné (čítače)
 - Monitory: vyšší konstrukty programovacího jazyka
Je možné semafor implementovat pomocí monitoru a naopak
- Smrtné objetí (deadlock)
- Odstranění sdílených zdrojů: zasílání zpráv
 - Synchronizace na úrovni zasílání a přijímání zpráv
 - Buffery

Procesy – plánování

- Sdílení (timesharing)
 - časové kvantum
 - přerušování
- 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)
 - Doplnková operace: změň rozsah alokované paměti (reallocate)
- Organizace paměti
- Čištění paměti (garbage collection)

Správa paměti OS

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

OS: paměť

- Většina paměti nevyužita
 - Zpracování cyklu (zbytek programu)
 - 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: Systém souborů

- Základní funkce:
 - Vytvoření souboru
 - Čtení a psaní z/do souboru
 - Odstranění (smazání) souboru
 - Spuštění souboru (soubor=program)
- Podpora na úrovni operačního systému

Struktura systému souborů

- Hierarchické systémy:
 - Kořen (root)
 - Adresáře jako speciální typ (meta)souboru: drží informace o souborech, nikoliv jejich vlastní data
- Databázové systémy:
 - Soubory (jejich části resp. jejich metadata) jako položka v databázi
 - Bohatší množina operací
 - Složitější implementace

Apache Hadoop má prvky databázového systému souborů

- Hadoop Distributed File System
- Soubory jsou rozděleny na části, které jsou distribuovány na prvky clusteru
- Primární přístup k souborům přes nativní rozhraní HDFS

Struktura souborů

- Posloupnost bytů – vnitřní struktura pro OS neznáma
- Posloupnost záznamů (records)
- Strom – každý uzel má vlastní klíč

- Výše uvedeny **příklady** struktury, ne všechny varianty

Typ a přístup

- Typy souborů (v UNIXovém OS)
 - Řádné: běžné soubory
 - Adresáře: udržení hierarchické struktury
 - Speciální: přístup ke konkrétnímu zařízení (`/dev/mouse`, `/dev/audio`, `/dev/lp`); speciální `/proc` systém
 - Blokové: náhodný přístup na základní úrovni (`/dev/hd`, `/dev/kmem`)
- Přístupové metody; příklady:
 - Sekvenční
 - Náhodný (random)
 - Indexsekvenční (není v běžném UNIXu)

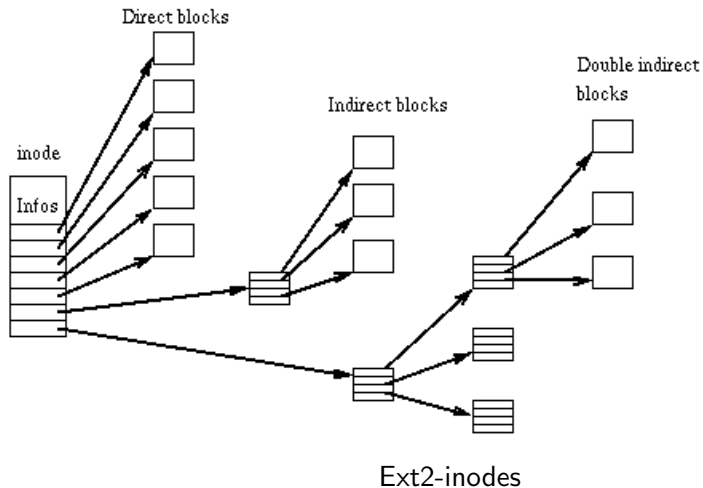
Struktura na disku

- Možné typy
 - Souvislé
 - souvislé posloupnost bloků (složitá alokace, plýtvání místem)
 - Provázaný seznam:
 - každý blok odkazuje na další (může růst, vyšší režie – pro ukazatel, složitý náhodný přístup)
 - Indexové:
 - Např. FAT (File Allocation Table) v MS DOSu
 - Tabulka pro všechny bloky na disku
 - Provázány odkazem na další blok daného souboru
 - inodes

Struktura – inodes

- Podobné indexové organizaci
- 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

inodes



Systémy souborů se žurnálováním

- Problém interní konzistence informací uvnitř systému souborů
 - co se stane při výpadku proudu či nečekaném zhroucení operačního systému
 - Riziko nekonzistentních dat při postupném zápisu
 - část dat nebo metadat není ještě zapsána
 - Klasické řešení: **fsck**
 - procházení všech datových struktur v systému souborů
 - nalezení a oprava nekonzistencí (zpravidla několik průchodů)
 - velmi pomalé pro velké systémy souborů
 - Alternativa: zapsat zvlášť prováděné změny a teprve poté je skutečně realizovat
 - úspěšný zápis všech dat a metadat vede ke smazání údajů
- journaled file system**
- V případě výpadku se použijí tato data na zajištění interní konzistence systému souborů

Volné bloky

- V tabulce
- Bitový vektor
- Provázaný seznam
- Většinou zpracovávají podle FCFS (First Come First Served)

Vyrovnávací paměť

- Obecně přístup pro skrytí *zpoždění* (latence)
- 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
 - Write-through: okamžitě po zápisu i na disk
 - Write-back: až po určité době (30 s)

OS: Přerušení

- Operační systémy obecně reagují na asynchronní události (events)
- *Přerušení*: mechanismus, jak přerušit vykonávanou práci na základě externí příčiny (nějaké události)

Význam přerušení

- Podpora I/O
- Problém v programovém vybavení
 - Neautorizovaný přístup
 - Nelegální instrukce nebo operandy
- Požadavek počítačem řízeného systému
- Zásah operátora
- Výpadek hardware

Příklady

- Přerušeni od časovače (přepínání procesů, multitasking, timeout, ...)
- Přerušeni od periferie (klávesnice, myš, síťová karta, ...)
- Přerušeni z procesoru (dělení nulou, chybná operace, ...)

OS: Principy přerušeni

- *Přerušit* běh aktuálního procesu
 - Nutno uložit stav
 - a zapamatovat místo návratu
- Více zdrojů a příčin přerušeni
 - Nutno rozlišit typy (příčinu) přerušeni
 - Nutno zapamatovat zdroj přerušeni

Obsluha přerušení

- Obsluha přerušení realizována v kernelu
 - Zajištění serializace
 - Bezpečnost
- Vyvolá tzv. přepnutí kontextu
- Multitasking fakticky není možný bez podpory přerušení

Další vlastnosti

- Maskování přerušení
 - dočasné a trvalé
 - možná ztráta přerušení/události
- Priorita přerušení/obsluhy
 - Základní tři úrovně:
 - Nemaskovaná přerušení: vyšší priorita
 - Aktuálně zpracovávané přerušení
 - Maskovaná přerušení: nižší priorita

Polling

- Polling = opakované dotazování (na stav/událost)
- Možná alternativa pro některá přerušení
 - Zaměstnává procesor
 - Může zůstat v uživatelském prostoru
- Příklad: neustálé dotazování se na zapsanou známku

OS: problém časování

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

OS časování: jiné řešení

```
Proces 1 {  
    Spust' diskovou operaci  
    Počkej na dokončení  
    Zpracuj získaná data  
}  
Proces 2 {  
    Nějaká jiná aplikace  
}
```

- Přehlednější
- OS musí „přepínat“ mezi procesy (*priorita*)

Programové vybavení – pohled dle použití

- 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

Programovací jazyky

- Rozlišujeme
 - Styl
 - Míru abstrakce
 - „Dialekt“

Programovací jazyky – styl

- Imperativní/Procedurální: C, Fortran, Pascal, Perl, Python
- Objektivě orientované: Java, C++, C#
- Deklarativní/Funkcionální: ML, Lisp, MIRANDA, Erlang
- Deklarativní/Logické: Prolog, GHC
- S jediným přiřazením: SISAL
- Produkční systémy: OPS5
- Sémantické sítě: NETL
- Neuronové sítě: SAIC ANSpec

Procedurální vs. deklarativní styl

```

fac := 1;
if n > 0 then
  for i:=1 to n do
    fac := i*fac;

```

```

|   fac(0)    := 1;
|   fac(n>0) := n*fac(n-1);
|
|-----
|
|   fac(0,1).
|   fac(N,F1*N) :- fac(N-1,F1).
|

```

Programovací jazyky – míra abstrakce

- Strojový jazyk: přímo kódy jednotlivých instrukcí
- Assembler: jména instrukcí, operandy, pojmenované cílové adresy skoků
- Vyšší jazyky: obecné konstrukty, tvoří „kontinuum“
 - Agregované datové typy
 - Cykly namísto skoků
 - Procedury a funkce
 - Procesy a vlákna

Programovací jazyky – implementace

- Překladače
 - Zdrojový kód–mezijazyk–strojový jazyk
 - Překlad a sestavení

Programovací jazyky – implementace

- Překladače
 - Zdrojový kód–mezijazyk–strojový jazyk
 - Překlad a sestavení
- Interprety
 - Abstraktní počítač
 - Vhodné pro složité operace (např. práce s texty, s maticemi a algebraickými objekty)

Programovací jazyky – implementace

- Překladače
 - Zdrojový kód–mezijazyk–strojový jazyk
 - Překlad a sestavení
- Interprety
 - Abstraktní počítač
 - Vhodné pro složité operace (např. práce s texty, s maticemi a algebraickými objekty)
- Just-in-time překladače (nejen Java)
 - Známý již od osmdesátých let (řešil se tak nedostatek paměti)