



PB001: Úvod do informačních technologií

Luděk Matyska (Eva Hladká)

podzim 2023





Obsah přednášky

Přerušeni

Programové vybavení

Klasifikace OS

Kernel operačního systému

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

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

OS: Principy přerušení

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

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

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, profily, ...
- Aplikační programy



Programovací jazyky

- Rozlišujeme
 - Styl
 - Míru abstrakce
 - „Dialekt“



Programovací jazyky – styl

- Imperativní/Procedurální: C, Fortran, Pascal, Perl, Python
- Objektivě orientované: Java, C++, C#, .NET
- 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;	fac(0) := 1;
if n > 0 then	fac(n>0) := n*fac(n-1);
for i:=1 to n do	
fac := i*fac;	
--	-----
	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)

Klasifikace OS

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

Monolitický 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 o'ag, ...)`
 - Čtení: `int read(int ↓ldes, char *buf, unsigned nbytes)`
 - Zápis: `int write(int ↓ldes, char *buf, unsigned nbytes)`
 - Zavření: `int close(int ↓ldes)`
- 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 periférií 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