

# B4B35OSY: Operační systémy

## Lekce 1. Úvod do operačních systémů

Petr Štěpán

stepan@fel.cvut.cz



21. září, 2022

# Outline

- 1 Úvod
  - Cíle předmětu
- 2 Malý návod na použití školy
- 3 Co je operační systém
- 4 OS (nejen osobního) počítače
- 5 Složení OS
- 6 Struktura OS
- 7 Ochrana jádra OS

# Obsah

- 1 Úvod
  - Cíle předmětu
- 2 Malý návod na použití školy
- 3 Co je operační systém
- 4 OS (nejen osobního) počítače
- 5 Složení OS
- 6 Struktura OS
- 7 Ochrana jádra OS

# B4B350SY – Operační systémy

Přednášející:

**Michal Sojka**, CIIRC

`Michal.Sojka@cvut.cz`

**Petr Štěpán**, FEL, katedra kybernetiky

`Stepan@fel.cvut.cz`

Komunikace:

- gitlab commits a issue
- MS teams
- e-mails

# Materiály

- Silberschatz A., Galvin P. B., Gagne G.: *Operating System Concepts*  
<http://codex.cs.yale.edu/avi/os-book/OS7/os7c/index.html>
- Tanenbaum, Andrew S a Albert S Woodhull: *Operating systems design and implementation*. 3rd ed. Upper Saddle River: Prentice-Hall, c2006, xvii, 1054 s. ISBN 0131429388
- <http://wiki.osdev.org/>
- <http://hypervisor.org/>
- YouTube lectures (anglicky):
  - CS 162 – UC Berkeley
  - OS-SP06 – Surendar Chandra – UC Berkeley
  - MIT 6.004

# Organizace předmětu

- Souhrnná podrobná literatura v češtině není
- Tyto prezentace (stránka předmětu  
<https://osy.pages.fel.cvut.cz/> dostupné přes  
<https://cw.fel.cvut.cz/wiki/courses/b4b35osy>)
- Cvičení částečně seminární, více samostatná práce, nutná domácí příprava
- Hodnocení:
  - Body ze cvičení - min. 25, max 65
    - Úlohy celkem až 50 bodů
    - Aktivita při hodině max 10 bodů
    - Nepovinné úlohy až 14 bodů
  - Písemná zkouška max 30 bodů
  - Ústní část max 10 bodů – dobrovolná (nutná pro A)

# Obsah

- 1 Úvod
  - Cíle předmětu
- 2 Malý návod na použití školy
- 3 Co je operační systém
- 4 OS (nejen osobního) počítače
- 5 Složení OS
- 6 Struktura OS
- 7 Ochrana jádra OS

# Cíle předmětu

Podle Hospodářských novin se Informatika vyučuje nejlépe na FEL, ČVUT (22. 1. 2015)

- OS patří k základům informatiky
- Poznat úkoly OS a principy práce OS
- Využívat OS efektivně a bezpečně
- Procvičit se v programování v C, hlavně v paralelizaci

Co NENÍ cílem tohoto předmětu

- Naučit Vás jak napsat aplikaci pod (X)Windows
- Naučit triky pro konkrétní OS
- Vytvořit OS – na to je málo času



# Proč studovat OS

- Pravděpodobně nikdo z vás nebude psát celý nový OS
- Proč tedy OS studovat?
  - Každý ho používá a jen málokdo ví jak pracuje
  - Jde o nejrozsáhlejší a nejsložitější IT systémy
  - Uplatňují se v nich mnohé různorodé oblasti
    - softwarové inženýrství,
    - netradiční struktury dat,
    - sítě, algoritmy, ...
  - Čas od času je potřeba OS upravit
    - pak je potřeba operačním systémům rozumět
    - psaní ovladačů, ...
    - Mnoho programátorských problémů lze na nižší úrovni vyřešit snadněji a efektivněji
  - Techniky užívané v OS lze uplatnit i v jiných oblastech
    - neobvyklé struktury dat, krizové rozhodování, problémy souběžnosti, správa zdrojů, ...
    - mnohdy aplikace technik z jiných disciplín (např. operační výzkum)
    - naopak techniky vyvinuté pro OS se uplatňují v jiných oblastech (např. při plánování aktivit v průmyslu)

# Naučit se lépe programovat

- Programování má různé podoby/úrovně (měli byste se seznámit se všemi):
  - Integrace high-level knihoven (mnohé webové a mobilní aplikace)
  - Aplikační programování (AI, počítačové hry, ...) – obsahují vlastní algoritmy
  - Nízko-úrovňové programování (OS, embedded systémy, ...) – pomezí SW a HW
- *"You might not think that programmers are artists, but programming is an extremely creative profession. It's logic-based creativity."*  
—John Romero

# Naučit se přehledně programovat

- *"Any fool can write code that a computer can understand. Good programmers write code that humans can understand."*  
—Martin Fowler
  - V jednoduchosti je krása
    - řešení úlohy továrna z loňského roku – na 200 ale i 2000 řádek kódu
  - V dnešní době je program způsob záznamu informací/znalostí
    - V jedné pekárně se porouchal stroj a museli povolat pekaře-důchodce, protože nikdo jiný neznal recept na chleba ani nerozuměl programu stroje.
  - Nepište kód pro počítač, ale pro lidi, aby byl srozumitelný a znalosti tam byly na první pohled viditelné
- *"Programming is the art of algorithm design and the craft of debugging errant code."*  
—Ellen Ullman

# Naučit se komentovat program

- Používejte vhodně pojmenované proměnné a funkce
- Komentujte co má funkce dělat a naznačte jak to dělá
- Komentujte jen to, co nelze vyjádřit programovacím jazykem
  - `i = 1; // do proměnné i přiřadíme hodnotu 1 – NE!`
- *“The cleaner and nicer the program, the faster it’s going to run. And if it doesn’t, it’ll be easy to make it fast.”*

—Joshua Bloch
- Opět – Nepište kód pro počítač, ale pro lidi, aby byl srozumitelný a znalosti tam byly na první pohled viditelné

# Obsah

- 1 Úvod
  - Cíle předmětu
- 2 Malý návod na použití školy
- 3 Co je operační systém
- 4 OS (nejen osobního) počítače
- 5 Složení OS
- 6 Struktura OS
- 7 Ochrana jádra OS

# Covid-19 a jiné katastrofy

- Výjimečný stav požaduje výjimečné výkony
- Větší nároky:
  - soustředění
  - iniciativu
  - domácí práci
  - organizace práce
- Řešení:
  - zvýšená komunikace, jak mezi Vámi tak směrem k nám
  - orientovat se v již probíhajících debatách

# Cíle vzdělávání

## ■ V obecné rovině

- Naučit kriticky myslet
- Naučit hledat zákonitosti

## ■ V konkrétní rovině

- Předat nějaké konkrétní znalosti (co je posix, cache, sběrnice)
- Předat nějaké konkrétní dovednosti (jak se programuje, jak efektivně vést projekt)

# O dobrém a špatném učení

- Povrchní přístup k učení
  - Úkoly dělám, abych splnil jejich zadání a dostal body
  - Výsledkem je zpravidla memorování
  
- Hlubkový přístup k učení
  - Úkoly dělám, abych splnil jejich účel
  - Výsledkem je zpravidla porozumění
  - Navíc je nutné najít účel úloh



# Proč porozumět a ne memorovat

- Schopnost spojit nové a dřívější znalosti
  - Pomáhá v chápání nových znalostí
  - Pomáhá odstranit chybné znalosti
- Schopnost použít znalosti
  - Znalosti lze spojit s každodenní zkušeností
- Schopnost uchovat znalosti
  - Dobře spojené a pochopené znalosti se pamatují déle
- Volba je na Vás !

# Co bude na přednáškách

- Výkladu se nedá uniknout
  - Náplň je většinou předem k dispozici
- Je to jako kino, ne? Návštěva kina je:
  - Pasivní zážitek s občas zajímavým příběhem
  - Nemusíte příliš přemýšlet
  - Desítky miliónů \$ vynaložené na udržení Vaší pozornosti
- Odkazy na další čtení k tématu jak (se) učit
  - S.Khan: Let's use video to reinvent education. TED(online)2011  
[https://www.ted.com/talks/sal\\_khan\\_let\\_s\\_use\\_video\\_to\\_reinvent\\_education](https://www.ted.com/talks/sal_khan_let_s_use_video_to_reinvent_education)
  - Obrácená výuka - flipped learning <https://flippedlearning.org/>

# Co bude na přednáškách

- Když chodíte na přednášky
  - očekává se, že se něco naučíte
- V čem je problém
  - Látka je složitá, ale při poslouchání to člověku nepříjde
  - Většina věcí se jeví logická – myšlenkové zkratky
  - Pro zvládnutí je nutné se jí nějakou dobu věnovat i po přednášce
    - ACM/IEEE CS Curriculum: na 1 hodinu přednášky v bakalářském studiu připadají 2–3 hodiny domácí přípravy

# Jak se něco na přednášce naučit

## ■ Neusnout

- Bez ohledu na to, jak těžké to může být
- Kdo spí, ten se nic nenaučí a přichází o souvislosti

## ■ Chodit pravidelně

- Nová látka staví na předchozích základech
- Naučíte se lépe rozumět přednášejícímu
- Když jsem minule nebyl, alespoň si přečíst přednášky

## ■ Aktivně poslouchat

- Nejlépe se nové věci naučíte při hledání vlastního vysvětlení, jak věci fungují
- Dává smysl to co slyšíte?
- Byli byste schopni to vysvětlit někomu, kdo na přednášce nebyl?

## ■ Pokud něco nedává smysl

- Zapište si co Vám nedává smysl
- Zkuste vymyslet otázku, jejíž zodpovědění by věci vyjasnilo a položte ji přednášejícímu

# Kdy a jak se ptát

- Když Vaše představa neodpovídá tomu co slyšíte
  - Nebo když Vám chybí část „skládanky“
  - Na konci přednášky byste měli být schopni položit několik otázek, alespoň upřesňujících
    - „Myslím si, že říkáte ..(vlastními slovy).., je to tak?“
- Než se zeptáte, zkuste si odpovědět
  - Pokud nevíte, nebo si nejste jisti, zeptejte se
- Při hledání otázek začnete pozorněji poslouchat
  - Začnete poslouchat s cílem se něco naučit
  - Naučíte se klást užitečné dotazy

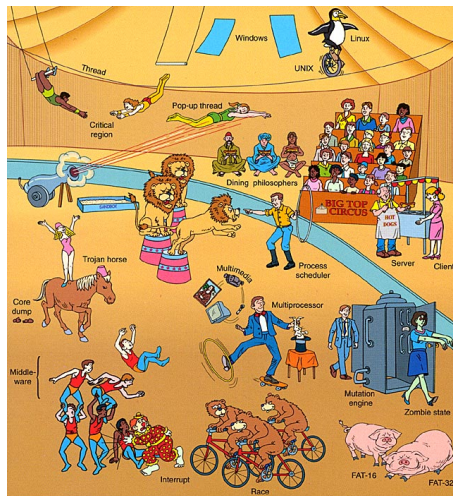
# Obsah

- 1 Úvod
  - Cíle předmětu
- 2 Malý návod na použití školy
- 3 Co je operační systém
- 4 OS (nejen osobního) počítače
- 5 Složení OS
- 6 Struktura OS
- 7 Ochrana jádra OS

# Co je operační systém

## Úkoly OS:

- Spouštět a dohlížet uživatelské programy
- Efektivní využití HW
- Usnadnit řešení uživatelských problémů
- Učinit počítač (snáze) použitelný
- Umíte použít počítač bez OS?



# Co je operační systém

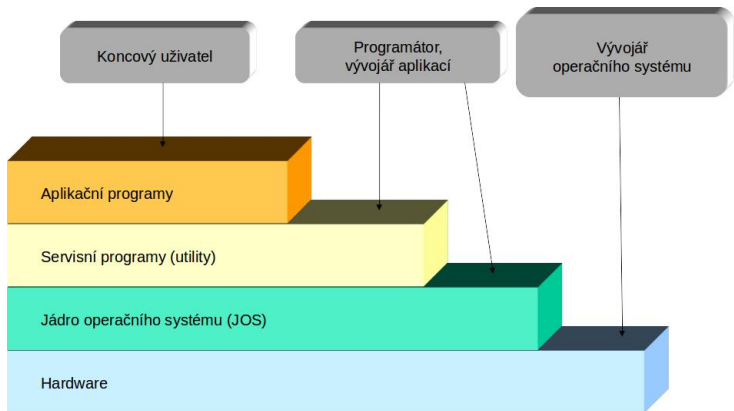
- Neexistuje žádná obecně platná definice
- Několik koncepcí pojmu OS
  - systémové (jen jádro a s ním související nadstavby)
  - „obchodní“ (to, co si koupíme pod označením OS)
  - organizační (včetně pravidel pro hladký chod systému)
- OS jako rozšíření počítače
  - Zakrývá komplikované detaily hardware
  - Poskytuje uživateli „virtuální stroj“, který se snáze ovládá a programuje
- OS jako správce systémových prostředků
  - Každý program dostává prostředky v čase
  - Každý program dostává potřebný prostor na potřebných prostředcích
  - Prostředky jsou CPU, paměť, periférie



# Co je operační systém

V této přednášce budeme brát operační systém jako jádro operačního systému

- ostatní (tzv. systémové) programy lze chápat jako nadstavbu jádra
- GUI – Windows je grafická nadstavba systémových programů



# Různorodost OS

- OS „střediskových“ (mainframe) počítačů – dnes již historický pojem
- OS superpočítačů (5 mil. jader, 200 PFlops, 13 MW příkon)
- OS datových a síťových serverů
- OS osobních počítačů a pracovních stanic
- OS reálného času (Real-time OS – řízení letadel, vlaků, raket, družic, apod.)
- OS přenosných zařízení – telefony, tablety
- Vestavěné OS (tiskárna, pračka, telefon, ...)
- OS čipových karet (smart card OS)
- ... a mnoho dalších specializovaných systémů

# Systémy reálného času – RT

- Nejčastěji řídicí zařízení v dedikovaných (vestavěných) aplikacích:
  - vědecký přístroj, diagnostický zobrazovací systém, systém řízení průmyslového procesu, monitorovací systémy
  - obvykle dobře definované pevné časové limity
  - někdy také subsystém univerzálního OS
- Klasifikace:
  - striktní RT systémy – Hard real-time systems
    - omezená nebo žádná vnější paměť, data se pamatují krátkodobě v RAM paměti
    - protipól univerzálních OS nepodporují striktní RT systémy
    - plánování musí respektovat požadavek ukončení kritického úkolu v rámci požadovaného časového intervalu
  - tolerantní RT systémy – Soft real-time systems
    - použití např. v průmyslovém řízení, v robotice
    - použitelné v aplikacích požadujících dostupnost některých vlastností obecných OS (multimedia, virtual reality, video-on-demand)
    - kritické úkoly mají přednost „před méně šťastnými“

# Více úloh současně – Multitasking

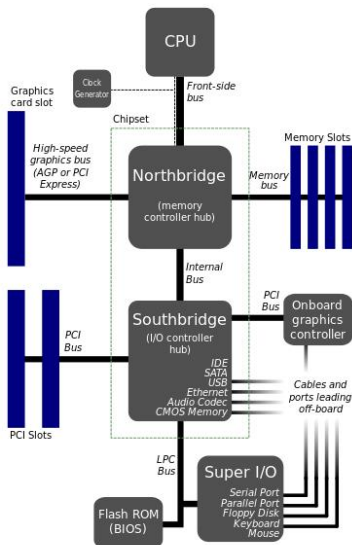
- Zdánlivé spuštění více procesů současně je nejčastěji implementováno metodou sdílení času tzv. Time-Sharing Systems (TSS)
- Multitasking vznikl jako nástroj pro efektivní řešení dávkového zpracování
- TSS rozšiřuje plánovací pravidla
  - o rychlé (spravedlivé, cyklické ) přepínání mezi procesy řešícími zakázky interaktivních uživatelů
- Podpora on-line komunikace mezi uživatelem a OS
  - původně v konfiguraci počítač – terminál
  - v současnosti v síťovém prostředí
- Systém je uživatelům dostupný on-line jak pro zpřístupňování dat tak i programů

# Obsah

- 1 Úvod
  - Cíle předmětu
- 2 Malý návod na použití školy
- 3 Co je operační systém
- 4 OS (nejen osobního) počítače**
- 5 Složení OS
- 6 Struktura OS
- 7 Ochrana jádra OS

# Osobní počítač

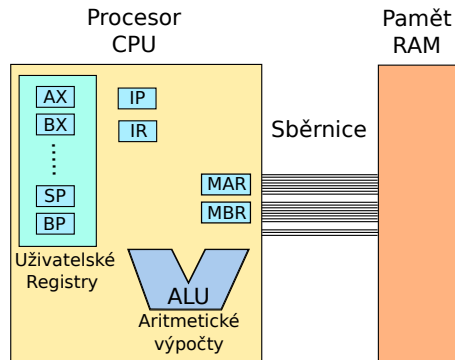
- Základem počítače je procesor – CPU
- Procesor je připojen sběrnicemi (bus, interconnect) k ostatním perifériím počítače – paměti, grafickému výstupu, disku, klávesnici, myši, síťovému rozhraní, atd.
- Činnost sběrnice řídí arbitr sběrnice



# Processor – CPU

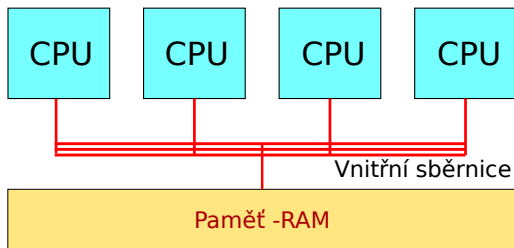
## ■ Základní vlastnosti:

- šířka datové a adresové sběrnice
- počet vnitřních registrů
- rychlost řídicího signálu – hodiny
- instrukční sada

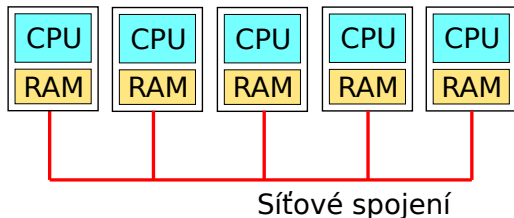


# Paralelní a distribuované systémy

Těsně vázaný  
multiprocessorový  
systém



Distribuovaný  
systém typu  
klient-server





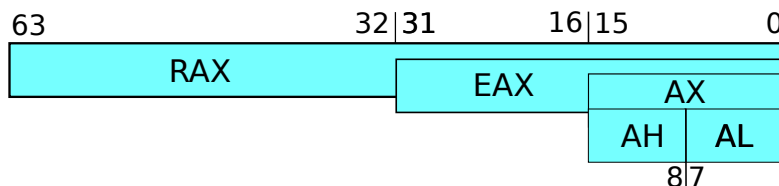
# Kvíz

Kolik jste si toho odnesli z APO?

- A - Vím úplně přesně jak pracuje procesor, paměť, sběrnice
- B - Vím zhruba, jak pracuje procesor, paměť, sběrnice
- C - Vím trochu, ale dost mi toho je nejasné
- D - Spíše nevím, jak pracuje procesor

# Processor – x86/AMD64

- Přehledný popis – [https://en.wikibooks.org/wiki/X86\\_Assembly](https://en.wikibooks.org/wiki/X86_Assembly)
- Všechny registry vzhledem ke zpětné kompatibilitě jsou 64/32/16/8 bitové

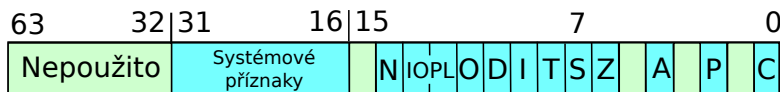


## Řídící a stavové registry

- EIP/RIP – instruction pointer – adresa zpracovávané instrukce
- EIR/RIR – instruction registr – kód zpracovávané instrukce
- EFLAGS/RFLAGS – stav procesoru povoleno/zakázáno přerušení, system/user mód, výsledek operace – přetečení, podtečení, rovnost 0, apod.

# Registr FLAGS

## RFLAGS registr



C – Carry flag

P – Parity flag

Z – Zero flag

S – Sign flag

O – Overflow flag

I – Interrupt enable

T – Trap flag

IOPL – I/O privilege level

A – Adjust flag

# Režimy práce procesoru

## FLAGS registr

- Dva režimy práce procesoru IOPL – základ hardwarových ochran
  - CPL0<sup>1</sup> = privilegovaný (systémový) režim
    - procesor může vše, čeho je schopen
  - CPL3 = uživatelský (aplikační) režim
    - privilegované operace jsou zakázány
- Privilegované operace
  - ovlivnění stavu celého systému (halt, reset, Interrupt Enable/Disable, modifikace Flags, modifikace registrů MMU )
  - instrukce pro vstup/výstup (in, out)
- Přechody mezi režimy
  - Po zapnutí stroje systémový režim
  - Přechod do uživatelského – modifikace Flags (popf nebo reti)
  - Přechod do systémového – pouze přerušení vč. programového

---

<sup>1</sup>Current privilege level

# Procesor – x86/AMD64

## Uživatelské registry

- programově dostupné registry pro ukládání hodnot programu *eax*, *ebx*, *ecx*, *edx*
- registry umožňující uchovat hodnotu, nebo ukazatel do paměti *esi*, *edi*, *ebp*
- *esp* – stack pointer – ukazatel zásobníku - detailněji dále
- AMD64/X86-64 přidává 8 dalších registrů *r8-r15*, ve formě *r8b* nejnižší bajt, *r8w* nejnižší slovo (16 bitů), *r8d* – nižších 32 bitů, *r8* – 64 bitový registr

# Kvíz

Jak znáte assembler x86?

- A - Všechny assembly jsou podobné, tedy tuším většinu instrukcí
- B - Umím jen RISC V, x86 netuším vůbec
- C - Sám jsem zkoumal programy na x86, RISC V bych už nedal
- D - Už si nepamatuji ani RISC V, ani x86

# Instrukce – x86/AMD64

## Instrukce “ulož hodnotu”

(běžně se používají dvě různé syntaxe pro zápis assembleru)

### AT&T

`movq zdroj 64b, cíl`

`movl zdroj 32b, cíl`

`movw zdroj 16b, cíl`

`movb zdroj 8b, cíl`

registry se značí `%ax`

hodnoty `$`, hex `0x`

`movl $0xff, %ebx`

### Intel

`mov cíl, zdroj`

pouze `ax`

číslo, hex postfix `h`

`mov ebx, 0ffh`

# Instrukce – x86/AMD64

Ulož hodnotu na adresu (odkaz do paměti)

AT&T

```
movl (%ecx), %eax
```

```
movl 3(%ebx), %eax
```

```
movl (%ebx, %ecx, 0x2), %eax
```

```
movl -0x20(%ebx, %ecx, 0x4), %eax
```

Intel

```
mov eax, [ecx]
```

```
mov eax, [ebx+3]
```

```
mov eax, [ebx+ecx*2h]
```

```
mov eax, [ebx+ecx*4h-20h]
```

- odkaz má 4 složky:  $\text{základ} + \text{index} * \text{velikost} + \text{posun}$
- pole struktur o velikosti *velikost*, *základ* je ukazatel na první prvek, *index* říká, který prvek chceme a *posun*, kterou položku uvnitř struktury potřebujeme.
- není potřeba použít všechny 4 složky



# Instrukce – x86/AMD64

## Aritmetika – AT&T syntax

operace co, k čemu

<code>addq \$0x05,%rax</code>	<code>rax = rax + 5</code>
<code>subl -4(%ebp), %eax</code>	<code>eax = eax - mem(ebp-4)</code>
<code>subl %eax, -4(%ebp)</code>	<code>mem(ebp-4) = mem(ebp-4)-eax</code>
<code>andX</code>	bitový and – argumenty typu X – b, w, l, q
<code>orX</code>	bitový or
<code>xorX</code>	bitový xor (nejrychlejší vynulování registru)
<code>mulX</code>	násobení čísel bez znamének
<code>divX</code>	dělení čísel bez znamének
<code>imulX</code>	násobení čísel se znaménky
<code>idivX</code>	dělení čísel se znaménky

# Instrukce – x86/AMD64

## Aritmetika s jedním operandem – AT&T syntax operace s cílím

<code>incl %eax</code>	<code>eax = eax + 1</code>
<code>decw (%ebx)</code>	<code>mem(ebx) = mem(ebx)-1</code>
<code>shlb \$3, %al</code>	<code>al = al«3</code>
<code>shrb \$1, %bl</code>	<code>bl=11000000, po bl=01100000</code>
<code>sarb \$1, %bl</code>	<code>bl=11000000, po bl=11100000</code>
<code>rorx, rorl</code>	bitová rotace doprava a doleva
<code>rcrx, rcl</code>	bitová rotace – přes C – carry flag

# Instrukce – x86/AMD64

## Podmíněné skoky

test a1, a2    tmp = a1 AND a2, Z tmp=0, C tmp<0

cmp a1, a2    tmp = a1-a2, Z tmp=0, C tmp<0

pak lze použít následující skoky

jmp kam        nepodmíněný skok, vlastně %eip=kam

je kam        jmp equal – skoč při rovnosti

jne kam        jmp not equal – skoč při nerovnosti

jg/ja kam      jmp greater – skoč pokud je a1 > a2 (sign/unsig)

jge/jae kam    skoč pokud je a1 >= a2 (sign/unsig)

jl/jb kam      jmp less – skoč pokud je a1 < a2 (sign/unsig)

jle/jbe kam    skoč pokud je a1 <= a2 (sign/unsig)

jz/jnz kam     skoč pokud je Z=1/0

jo/jno kam     skoč pokud je O (overflow) = 1/0

# Assembler v C programu

```
#include <stdio.h>
int main() {
    int c_a=-15, c_b=-5;
    asm volatile (
        "mov  %%eax, %%edx;"
        "sar  $0x1f, %%edx;"
        "idivl %%ebx;"
        : "+a" (c_a) : "b" (c_b) : "edx");
    printf ("-15/(-5)?=%i\n", c_a);

    c_a = -15; c_b = -5;
    asm volatile (
        "xor  %%edx, %%edx;"
        "divl %%ebx;"
        : "+a" (c_a) : "b" (c_b) : "edx");
    printf ("-15/(-5)?=%i\n", c_a);
```

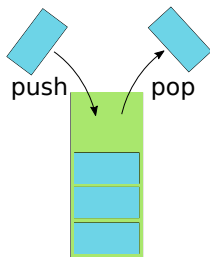
```
c_a = -15;
c_b = -5;
asm volatile (
    "mov  %%eax, %%edx;"
    "sar  $0x1f, %%edx;"
    "divl  %%ebx;"
    : "+a" (c_a) : "b" (c_b) : "edx");
printf ("-15/(-5)?=%i\n", c_a);

return 0;
}
```

# Zásobník

## Zásobník:

- obecná struktura LIFO
- operace push vloží data do zásobníku
- operace pop vybere data ze zásobníku



## Implementace:

- implementace registrem *SP* - ukazuje na vrchol zásobníku
- konvence - při každém pop se zvětšuje registr *SP* o velikost operandu, při push se *SP* zmenšuje.

<code>pushl %eax</code>	ulož <code>eax</code> na zásobník
<code>popw %bx</code>	vyber ze zásobníku 2 bajty do <code>bx</code>
<code>pushf / popf</code>	ulož/vyber register EFLAGS
<code>pusha / popa</code>	ulož/vyber všechny uživatelské registry

# Zásobník - Kvíz

```
#include <stdio.h>
#include <unistd.h>
int main();

int b() {
    printf ("Co tu delam?\n");
    _exit (1);
}

void f(int x) {
    unsigned long local[2], sp;
    int i;

    local[0]=1; local[1]=2;
    for (i=9; i>=0; i--) {
        printf ("%i - %016lx\n", i, local[i]);
    }
    printf ("main %p\n", &main);
    asm volatile (
        "mov %%rsp, %%rax;"
        : "=a" (sp) : : );
    printf ("rsp %016lx\n", sp);
    local[5]=( unsigned long)&b;
}

int main() {
    f(10);
    printf ("Proc?\n");
    return 0;
}
```

Vlevo vidíte program stack-64.c.  
Můžete si ho stáhnout ze stránek  
předmětu v balíčku příklady k první  
přednášce a zkompileovat pomocí:  
make stack-64

Co vytiskne na konec tento  
program?

- A - Proc?
- B - Co tu delam?
- C - Nic - program spadne
- D - Nic - program se normálně ukončí

# Zásobník - řešení

```
#include <stdio.h>
#include <unistd.h>
int main();

int b() {
    printf ("Co tu delam?\n");
    _exit(1);
}

void f(int x) {
    unsigned long local[2], sp;
    int i;

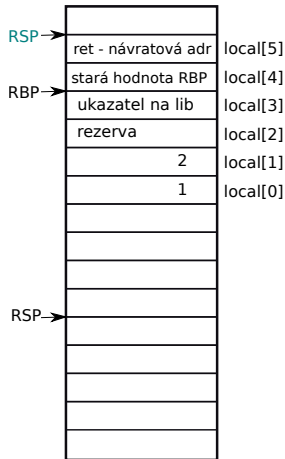
    local[0]=1; local[1]=2;
    for (i=9; i>=0; i--) {
        printf ("%i - %016lx\n", i, local[i]);
    }
    printf ("main %p\n", &main);
    asm volatile (
        "mov %%rsp, %%rax;"
        : "=a" (sp) : : );
    printf ("rsp %016lx\n", sp);
    local[5]=( unsigned long)&b;
}

int main() {
    f(10);
    printf ("Proc?\n");
    return 0;
}
```

```
push    %rbp
mov     %rsp,%rbp
sub     $0x40,%rsp
```

```
leaveq
retq
```

```
mov     $0xa, %edi
call    11c7 <f>
```



# Funkce zásobníku

## Zásobník:

- parametry pro funkci
- kam se vrátit po ukončení funkce, místo odkud program volal funkci
- lokální proměnné funkce
  - zásobník je většinou malý
  - omezená velikost lokálních proměnných
  - pozor při rekurzi - lépe se rekurzi vyhnout



# Instrukce – x86/AMD64

## Volání funkce

`call adr`     vlastně `push %eip, jmp adr`

`ret`           vlastně `pop %eip`

`leave`        vlastně `mov %ebp, %esp, pop %ebp`

Lokální proměnné ve funkci – příklad implementace

**`push %ebp`**            ; Uložíme hodnotu EBP do zásobníku

**`mov %esp, %ebp`** ; Zkopírujeme hodnotu registru ESP to EBP

**`sub $12, %esp`** ; Snizíme ukazatel zásobníku o 3x4 bajty

První proměnná bude na adrese  $-4(\%ebp)$ , druhá  $-8(\%ebp)$

První parametr bude na adrese  $8(\%ebp)$ , další  $12(\%ebp)$

**`mov %ebp, %esp`** ; Vratíme ukazatel zpět na původní pozici.

**`pop %ebp`**            ; Obnovíme původní hodnotu registru EBP

**`ret`**                 ; Navrát z funkce

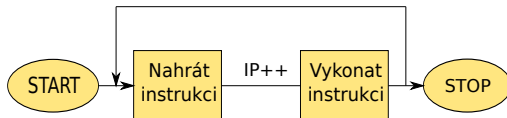
# Instrukce – x86/AMD64

## Složitost assembleru

- Algoritmus se dá přeložit různými způsoby do assembleru
- Různé způsoby pracují různě rychle a jsou rozdílně dlouhé a rozdílně přehledné
- `xor %ebx, %ebx` je to samé jako `mov $0, %ebx`
- `lea adresa, registr` – load effective address – nastaví hodnotu ukazatele do zadaného registru
- `lea -12(%esp), %esp` je to samé jako `sub $12, %esp`
- `lea` je výhodnější vzhledem k předzpracování instrukcí, nezatěžuje ALU jednotku (ovšem třeba Atom má zpracování adr. pomalejší než ALU).

# Pracovní krok procesoru

- Procesor pracuje v krocích.
  - Pipelining a superskalární procesory se navenek tváří, že pracují stejným způsobem
    - Přinejhorším jsou schopny prohodit pořadí vykonání dvou instrukcí
- Jeden krok obsahuje fáze:
  - Přípravná fáze (fetch cycle)
    - nahrává do procesoru instrukci podle IP a umístí její kód do IR
    - na jejím konci se inkrementuje IP
  - Výkonná fáze (execute cycle)
    - vlastní provedení instrukce
    - může se dále obracet (i několikrát) k paměti



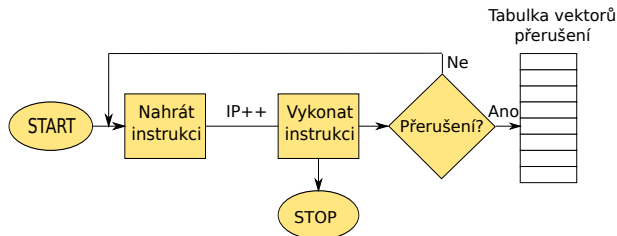
```

loop:  FETCH;      /* z adresy IP nahraj data do IR */
        Increment(IP);
        EXECUTE;   /* provede operaci uloženou v IR */
end loop
  
```

# Přerušení (výjimky)

- Přerušení normální posloupnosti provádění instrukcí
  - cílem je zlepšení účinnosti práce systému
  - je potřeba provést jinou posloupnost příkazů jako reakci na nějakou „neobvyklou“ externí událost
  - přerušující událost způsobí, že se pozastaví běh aktuálně vykonávaného programu v CPU takovým způsobem, aby ho bylo možné později znovu obnovit, aniž by to přerušovaný program „poznal“
- Souběh I/O operace
  - přerušení umožní, aby po začátku přenosu dat z/do periférie CPU prováděla jiné akce než čekání na konec I/O operace
  - činnost CPU se později přeruší iniciativou „I/O modulu“
  - CPU předá řízení na obslužnou rutinu přerušení (Interrupt Service Routine) – standardní součást OS
- CPU testuje nutnost věnovat se obsluze přerušení alespoň po dokončení každé instrukce
  - existují výjimky (např. „blokové instrukce“ Intel)

# Pracovní krok s přerušením



INTF=False; /\* vymaz preruseni \*/

loop: FETCH;

Increment(IP);

EXECUTE;

IF povoleno preruseni && INTF then

Uloz FLAGS na zasobnik

Uloz IP na zasobnik

FLAGS nastav CPL0 a zakaz preruseni

IP = vektoru preruseni

end loop

# Obsluha přerušení

- Žádost se vyhodnotí na přípustnost (priority přerušení)
- Procesor přejde do zvláštního cyklu
  - FLAGS se uloží na zásobník (registr FLAGS se mění již při vstupu do přerušení a také většina instrukcí mění hodnotu FLAGS; je tedy nutné ho uložit co nejdříve).
  - Na zásobník se uloží i hodnota čítače instrukcí IP (návrátová hodnota z přerušení).
  - Do FLAGS se vygeneruje nové stavové slovo s nastaveným CPL0. Nyní je CPU v privilegovaném režimu
  - IP se nahradí hodnotou z vektoru přerušení – skok na obsluhu přerušení
- Procesor přechází do normálního režimu práce a zpracovává obslužnou rutinu přerušení v privilegovaném módu
  - Obslužná rutina musí být transparentní, tj. programově se musí uložit všechny registry CPU, které obslužná rutina použije, a před návratem z přerušení se opět vše musí obnovit tak, aby přerušená posloupnost instrukcí nepoznala, že byla přerušena.
  - Obslužnou rutinu končí instrukce „návrat z přerušení“ IRET mající opačný efekt: z vrcholu zásobníku vezme položky, které umístí zpět do IP a FLAGS

# Druhy přerušení (x86)

- Každé přerušení má své číslo odkazující do tabulky přerušení, kde je tzv. vektor přerušení
- Vektor přerušení obsahuje adresu programu, od které se začne vykonávat kód při výskytu daného přerušení
- Přerušení se dělí vzhledem k vykonávanému programu na synchronní a asynchronní

## Synchronní přerušení

- Chyba dělení (dělení nulou) 0
- Program break 3
- Chybná instrukce 6
- Chybějící segment 11
- Chyba segmentu zásobníku 12
- Chyba ochrany 13
- Chyba stránky 14

## Asynchronní přerušení

- Nemaskovatelné přerušení 2
- časovač 32
- uživatelské přerušení 32–255  
(síťová karta, klávesnice, ...)

# Zdroje přerušení

- Vnitřní přerušení – problém při zpracování strojové instrukce
  - instrukce nebo data nejsou v paměti - chyba stránky, chyba segmentu
  - instrukci nelze provést - dělení nulou, ochrana paměti, nelegální instrukce
  - nutno reagovat okamžitě, nelze dokončit instrukci, někdy nelze ani načíst instrukci
- Vnější přerušení – vstupně/výstupní zařízení
  - asynchronní s během procesoru
  - signalizace potřeby reagovat na vstup/výstup
  - reakce po dokončení vykonávané instrukce
- Programové přerušení – strojová instrukce proved' přerušení
  - využívá se k ochraně jádra OS
  - obsluha přerušení může používat privilegované instrukce
  - lze spustit pouze kód připravený OS



# Vícenásobné přerušení

## ■ Sekvenční zpracování

- během obsluhy jednoho přerušení se další požadavky nepřijímají (pozdržují se, IF bit v registru FLAGS)
- jednoduché, ale nevhodné pro časově kritické akce

## ■ Vnořené zpracování

- prioritní mechanismus
- přijímají se přerušení s prioritou striktně vyšší, než je priorita obsluhovaného přerušení

## ■ Odložené zpracování

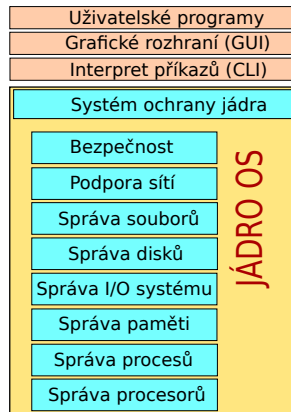
- V přerušení se provede pouze nejnutnější obsluha zařízení, zbytek se provede později mimo přerušení (deferred jobs, workqueues, ...)
- Neblokují se zbytečně další přerušení

# Obsah

- 1 Úvod
  - Cíle předmětu
- 2 Malý návod na použití školy
- 3 Co je operační systém
- 4 OS (nejen osobního) počítače
- 5 Složení OS**
- 6 Struktura OS
- 7 Ochrana jádra OS

# Složky OS

- Správa procesorů
- Správa procesů
- Správa (hlavní, vnitřní) paměti
- Správa I/O systému
- Správa disků – vnější (sekundární) paměti
- Správa souborů
- Podpora sítí
- Bezpečnost - security
- Systém ochrany jádra



# Interpret příkazů

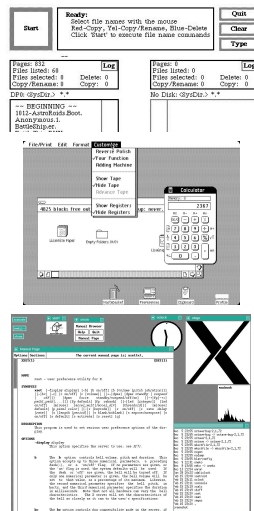
- Většina zadání uživatele je předávána operačnímu systému řídicími příkazy, které zadávají požadavky na
  - správu a vytváření procesů
  - ovládání I/O
  - správu sekundárních pamětí
  - správu hlavní paměti
  - zpřístupňování souborů
  - komunikaci mezi procesy
  - práci v síti, ...
- Program, který čte a interpretuje řídicí příkazy se označuje v různých OS různými názvy
  - Command-line interpreter (CLI), shell, cmd.exe, sh, bash, ...
  - Většinou rozumí jazyku pro programování dávek (tzv. skriptů)
  - Interpret příkazů není částí jádra OS
  - Interpret příkazů pracuje v uživatelském režimu, který je stejný jako pro Vaše programy

# Systémové nástroje

- Poskytují prostředí pro vývoj a provádění programů
- Typická skladba
  - Práce se soubory, editace, kopírování, katalogizace, ...
  - Získávání, definování a údržba systémových informací
  - Modifikace souborů
  - Podpora prostředí pro různé programovací jazyky
  - Sestavování programů
  - Komunikace
  - Anti-virové programy
  - Šifrování a bezpečnost
  - Aplikační programy z různých oblastí
- Systémové nástroje pracují v uživatelském režimu, který je stejný jako pro Vaše programy

# GUI

- První Xerox Alto (1973)
- Apple Lisa (1983)
- X window (1984) – MIT, možnost vzdáleného terminálu přes síť
- Windows 1.0 pro DOS (1985)
- Windows 3.1 (1992) podpora 32-bitových procesorů s ochranou paměti, vylepšená grafika
- Windows NT (1993) – preemptivní multitasking, předchůdce Windows XP (2001)



# Jádro OS

- Poskytuje ochranu/izolaci
  - Aplikačních programů mezi sebou
  - Hardwaru před škodlivými aplikacemi
  - Dat (souborů) před neoprávněnou manipulací
- Řídí přidělování zdrojů aplikacím
  - Paměť, procesorový čas, přístup k HW, síti, ...
- Poskytuje aplikacím služby
  - Jaké?

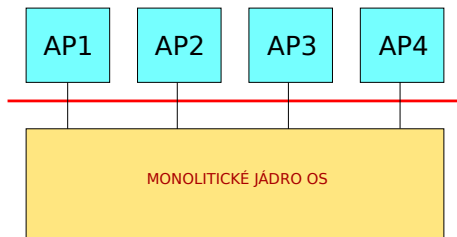
# Obsah

- 1 Úvod
  - Cíle předmětu
- 2 Malý návod na použití školy
- 3 Co je operační systém
- 4 OS (nejen osobního) počítače
- 5 Složení OS
- 6 Struktura OS**
- 7 Ochrana jádra OS



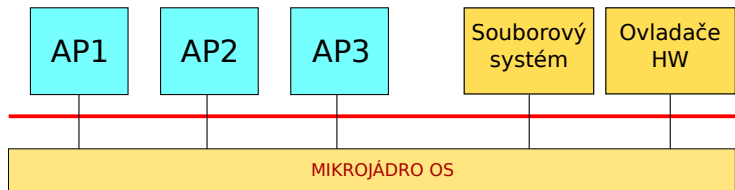
# Vykonání služeb jádra OS

- Klasický monolitický OS
  - Non-process Kernel OS
  - Procesy – jen uživatelské a systémové programy
  - Jádro OS je prováděno jako monolitický (byť velmi složitý) program v privilegovaném režimu
    - „USB MIDI má přístup ke klíči k šifrování disku :-)” CVE-2016-2384
- Služba jádra OS je typicky implementována jako kód v jádře, běžící jako přerušení využívající paměťový prostor volajícího programu

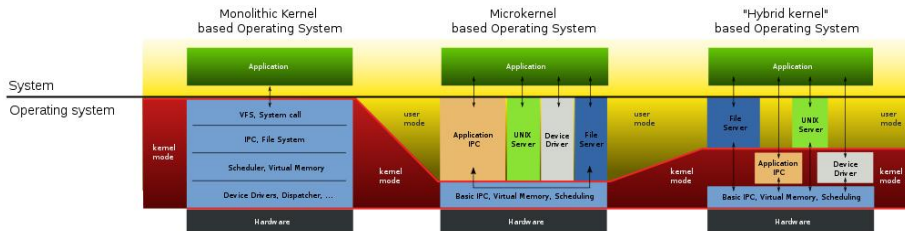


# Procesově orientované jádro OS

- OS je soustavou systémových procesů
- Funkcí jádra je tyto procesy separovat, ale umožnit přitom jejich kooperaci
  - Minimum funkcí je potřeba dělat v privilegovaném režimu
  - Jádro pouze ústředna pro přepojování zpráv
  - Řešení snadno implementovatelné i na multiprocesech
- Malé jádro  $\Rightarrow$  mikrojádro ( $\mu$ -jádro) – (microkernel)



# Porovnání JOS

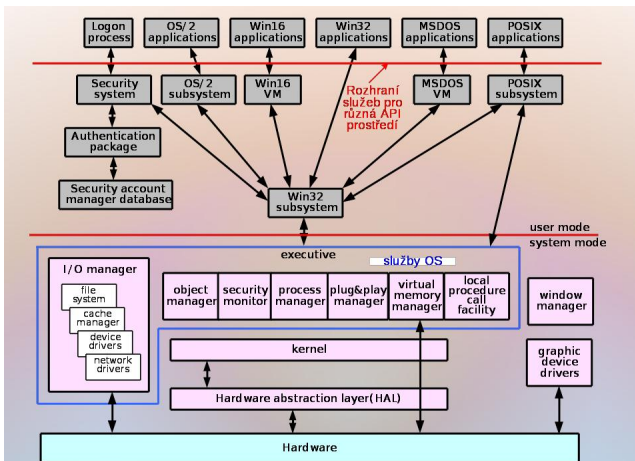


# Mikrojádro – vlastnosti

- OS se snáze přenáší na nové hardwarové architektury,
  - $\mu$ -jádro je malé
- Vyšší spolehlivost – modulární řešení
  - moduly jsou snáze testovatelné
- Vyšší bezpečnost
  - méně kódu se běží v privilegovaném režimu
- Pružnější, snáze rozšiřitelné řešení
  - snadné doplňování nových služeb a rušení nepotřebných
- Služby jsou poskytovány unifikovaně
  - výměnou zpráv
- Přenositelné řešení
  - při implementaci na novou hardwarovou platformu stačí změnit  $\mu$ -jádro
- Podpora distribuovanosti
  - výměna zpráv je implementována v síti i uvnitř systému
- Podpora objektově-orientovaného přístupu
  - snáze definovatelná rozhraní mezi aplikacemi a  $\mu$ -jádrem
- To vše za cenu
  - zvýšené režie, volání služeb je nahrazeno výměnou zpráv mezi aplikačními a systémovými procesy

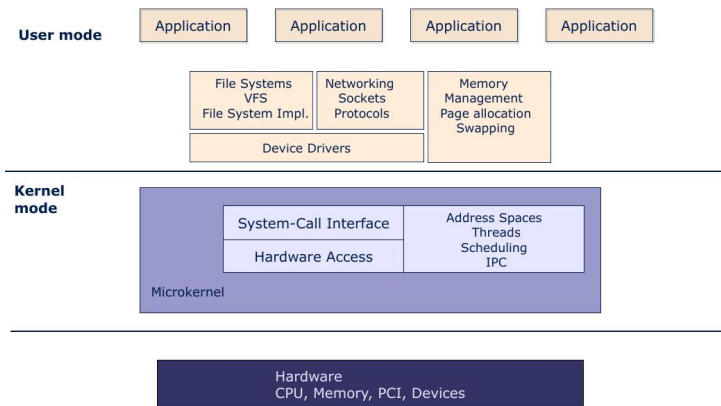
# Windows (XP)

JOS Windows má architekturu  $\mu$ -jádra, ale vše běží v jednom adresním prostoru, takže se jedná o monolitické jádro<sup>2</sup>.



<sup>2</sup><https://techcommunity.microsoft.com/t5/Windows-Kernel-Internals/One-Windows-Kernel/ba-p/267142>

# L4Re – OS se skutečným $\mu$ -jádre



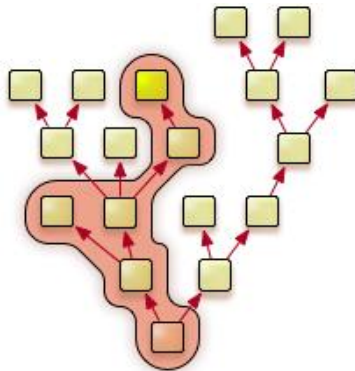
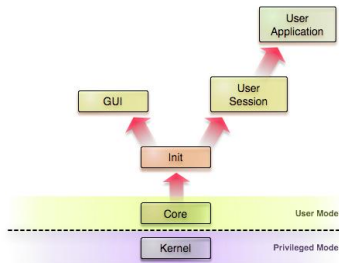
<http://os.inf.tu-dresden.de>

<http://www.kernkonzept.com/>

# Genode – OS se skutečným $\mu$ -jádre

Jeden z cílů: Omezit velikost “Trustued computing base”

<http://genode.org/>



# NOVA – $\mu$ -jádro

## Systémová volání OS NOVA:

- call
- reply
- create\_pd
- create\_ec
- create\_sc
- create\_pt
- create\_sm
- revoke
- lookup
- ec\_ctrl
- sc\_ctrl
- pt\_ctrl
- sm\_ctrl
- assign\_pci
- assign\_gsi

Výukový OS – bude používán na cvičení

- Víc systémových volání opravdu nemá
- PD – protection domain – proces
- EC – execution context
- SC – scheduling context
- PT – portal
- SM – semafor

My si na cvičení další systémová volání doprogramujeme, abyste viděli, jak se to dělá.



# Složitost jádra OS

OS mohou (ale nemusí) být funkčně velmi složité

OS	Rok	# služeb
Unix	1971	33
Unix	1979	47
Sun OS4.1	1989	171
4.3 BSD	1991	136
Sun OS4.5	1992	219
Sun OS5.6 (Solaris)	1997	190
WinNT 4.0	1997	3443
Linux 2.0	1998	229
Linux 4.4	2016	332
NOVA	2014	15

Počty cyklů CPU spotřebovaných ve WinXP při

- Zaslání zprávy mezi procesy: 6K–120K (dle použité metody)
- Vytvoření procesu: 3M
- Vytvoření vlákna: 100K
- Vytvoření souboru: 60K
- Vytvoření semaforu: 10K–30K
- Nahrání DLL knihovny: 3M
- Obsluha přerušení/výjimky: 100K–2M
- Přístup do systémové databáze (Registry) : 20K

Počty cyklů CPU spotřebovaných v OS NOVA při

- Zaslání zprávy mezi procesy: 300–600 (dle použité metody)

# Složitost jádra OS

OS jsou velmi rozsáhlé

Údaje jsou jen orientační, Microsoft data nezveřejňuje  
SLoC (Source Lines of Code) je velmi nepřesný údaj: Tentýž  
programový příkaz lze napsat na jediný nebo celou řadu řádků.

OS	Rok	SLoC
Windows 3.1	1992	3mil.
Windows NT 3.5	1993	4mil.
Windows 95	1995	15mil.
Windows NT 4.0	1997	16mil.
Windows 98 SR-2	1999	18mil.
Windows 2000 SP5	2002	30mil.
Windows XP SP2	2005	48mil.
Windows 7	2010	není známo
Linux 4.13 (jen JOS)	2017	16.8mil.
NOVA	2014	10tis.

# Obsah

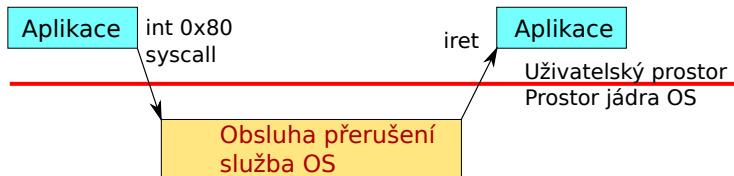
- 1 Úvod
  - Cíle předmětu
- 2 Malý návod na použití školy
- 3 Co je operační systém
- 4 OS (nejen osobního) počítače
- 5 Složení OS
- 6 Struktura OS
- 7 Ochrana jádra OS**

# Ochrana jádra OS

- Ochrana jádra
  - mechanismus pro kontrolu a řízení přístupu k systémovým a uživatelským zdrojům (paměť, HW zařízení, soubory, ...)
- Systém ochran „prorůstá“ všechny vrstvy OS
- Systém ochran musí
  - rozlišovat mezi autorizovaným a neautorizovaným použitím
  - poskytnout prostředky pro prosazení legální práce
- Detekce chyb
  - Chyby interního a externího hardware
    - Chyby paměti, výpadek napájení
    - Chyby na vstupně/výstupních zařízeních či mediích („díra“ na disku)
  - Softwarové chyby
    - Aritmetické přetečení, dělení nulou
    - Pokus o přístup k „zakázaným“ paměťovým lokacím (ochrana paměti)
  - OS nemůže obsloužit žádost aplikačního programu o službu
    - Např. „k požadovanému souboru nemáš právo přistupovat“

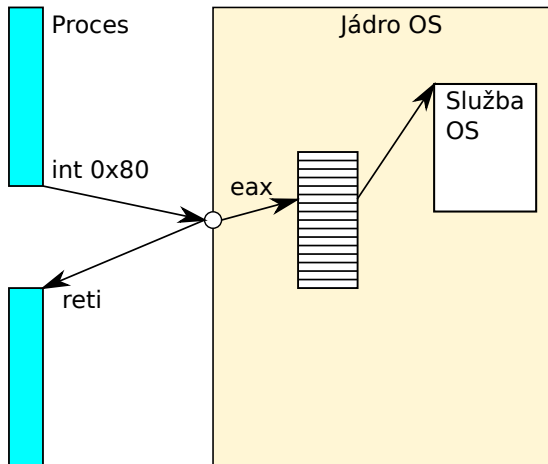
# Ochrana jádra OS

- Základ ochrany OS, přechod do systémového módu
  - Intel x86 rozlišuje 4 úrovně ochrany (privilege level): 0 – jádro OS, 3 – uživatelský mód
  - Jiné architektury mají většinou jen dva módy (jeden bit ve stavovém slově)
  - V uživatelském módu jsou některé instrukce zakázány (opakování – jaké?)
- Přechod z uživatelského módu do systémového
  - pouze programově vyvolaným přerušením
  - speciální instrukce (trap, int, sysenter, swi, ...)
  - nejde spustit cokoliv, spustí se pouze kód připravený operačním systémem
  - Systémová volání – služby jádra (system calls)
- Přechod ze systémového módu do uživatelského:
  - Speciální instrukce či nastavení odpovídajících bitů ve stavovém slově FLAGS
  - Návrat z přerušení



# Ochrana jádra OS

- Uživatel má do jádra OS přístup pouze přes obsluhu přerušení



# Kdy vlastně OS běží?

Jádro OS běží když:

- nastane přerušení nebo vyjímka
- uživatelský program zavolá službu OS

Jindy neběží?

Ještě na začátku spuštění počítače, připraví vše pro běh procesů a spustí první proces. Pak už jen čeká na přerušení, vyjímky a systémová volání.

# Rozdíl mezi root a jádrem OS

Root je sice správce systému, ale jedná se jen o obyčejné procesy v uživatelském procesu, které mají více práv, ale nemohou přistupovat k HW. I root proces musí využívat systémové služby ke komunikaci s HW.

Jádro OS není proces, jedná se o mnoho funkcí spouštěných přerušeními, výjimkami a systémovými voláními uživatelských procesů. Jádro OS může dělat úplně vše, co může počítač vykonat.