Operační systémy 1

Úvod do operačních systémů

Petr Krajča

Katedra informatiky
Univerzita Palackého v Olomouci

Organizační informace

- email: petr.krajca@upol.cz
- přednáška/konzultace: pátek 11:30 – 13:00
- cvičení
- konzultační hodiny
  - úterý 12:00 – 13:00
  - alternativně emailem nebo po předchozí domluvě
- www: `http://phoenix.inf.upol.cz/~krajcap/courses/2025LS/XOS1/`
- rozšiřující texty a praktické úkoly (nepovinné, ale užitečné)
- **není možné pořizovat obrazové a zvukové záznamy přednášek bez předchozího souhlasu vyučujícího**

Literatura

- Keprt A. *Operační systémy. 2008*
- Keprt A. *Assembler. 2008*
- Silberschatz A., Galvin P.B., Gagne G. *Operating System Concepts, 7th Edition.* John Wiley & sons, 2005. ISBN 0-471-69466-5.
- Tanenbaum A.S. *Modern Operating Systems, 2nd ed.* Prentice-Hall, 2001. ISBN 0-13-031358-0.
- Stallings, W. *Operating System Internals and Design Principles, Fifth Edition.* Prentice Hall, 2004. ISBN 0-13-127837-1.
- Bovet D., Cesati M. *Understanding Linux Kernel, 3rd ed.* O'Reilly, 2006. ISBN 0596005652.
- Solomon D.A., Russinovich M. E. *Windows Internals: Covering Windows Server 2008 and Windows Vista.* Microsoft Press, 2009. ISBN 0735625301.
- Jelínek L. *Jádro systému Linux: kompletní průvodce programátora.* Brno, Computer Press, 2008.

Operační systém

- **abstrakce HW**
- vyvíjet software na míru jednoho HW náročné/neefektivní (obvykle); hardware je neuvěřitelně složitý
- operační systém – rozhraní mezi HW a SW
- **operační systém poskytuje abstrakci nad daným hardwarem** (+ jazyky vyšší úrovně)
- v konečném důsledku několik úrovní abstrakce

Operační systém

Vrstvy HW/SW

1. hardware
2. operační systém (OS)
3. standardní knihovna (libc, CRT)
4. systémové nástroje
5. aplikace

- jádro OS vs. aplikace
- hranice mezi vrstvami nemusí být ostré
- situace se komplikuje – virtualizace, běhová prostředí
- další funkce: **operační systém zajišťuje správu zdrojů** – sdílení času (CPU, zařízení), místa (paměť, disky)

Historie operačních systémů

- 1. generace (1945 – 1955): relé, elektronky a program „zadrátovan" do počítače
- 2. generace (1955 – 1965): tranzistory, děrné štítky, dávkové zpracování a FORTRAN
- 3. generace (1965 – 1980): integrované obvody, IBM System/360 a minipočítače PDP
  - multitasking
  - timesharing (CTSS – MIT)
  - současná práce více uživatelů, ale pořád prvky dávkového zpracování
  - spooling (sdílení periferií)
  - virtuální paměť; první sítě
- 4. generace (1980 – současnost): vysoký stupeň integrace; Intel 8080, x86; CP/M, DOS, Windows 95/NT, Unix, GNU/Linux
- 5. generace (1990 – současnost): mobilní OS; Android, iOS

Typy operačních systémů

Podle určení

- Mainframy – OS/400, zOS
- Serverové/Multiprocesorové – *BSD, AIX, GNU/Linux, HP-UX, macOS (Darwin), Solaris, Windows NT, ...
- Desktopové – *BSD, GNU/Linux, macOS, Windows NT, ...
- Realtime – VxWorks, QNX, ...
- Distribuované
- Mobilní zařízení – Android, Bada, Blackberry OS, iOS, Symbian, Windows Phone, WebOS, Windows 10 Mobile, ...
- Experimentální/výukové – Minix, Plan 9

Historické záležitosti

- CP/M, MS-DOS, Windows 9x
- BeOS, Mac OS (classic), OS/2

Architektura počítače

John von Neumannova architektura

- CPU (ALU, **řadič**)
- paměť **společná pro program i data** (vs. harvardská architektura)
- vstup/výstup
- sběrnice (řídící, adresní, datová)
- instrukce procesoru jsou zpracovávány v řadě za sebou (není-li uvedeno jinak)

CPU (1/2)

Obecná struktura CPU

- Aritmeticko-logická jednotka (ALU) – provádí výpočty
- řídící jednotka – řídí chod CPU
- registry – slouží k uchování právě zpracovávaných dat (násobně rychlejší přístup než do paměti); speciální registry obsluhující chod CPU: IP (instruction pointer), program status word (PSW, FLAGS), IR (instruction register), SP (stack pointer)

Instrukční sada (ISA)

- sada instrukcí ovládající procesor (specifická pro daný CPU/rodinu CPU)
- instrukce a jejich operandy jsou reprezentovány jako čísla $\Longrightarrow$ strojový kód
- každá instrukce má obvykle 0 až 3 operandy (může to být registr, konstanta nebo adresa místa v paměti)
- pro snazší porozumění se instrukce CPU zapisují v jazyce symbolických adres (assembleru)

Výpočet faktoriálu: Intel x86

```
00000000 <main>:
 0:   8b 4c 24 04             mov    ecx,DWORD PTR [esp+0x4]
 4:   b8 01 00 00 00          mov    eax,0x1
 9:   83 f9 00                cmp    ecx,0x0
 c:   0f 8e 0a 00 00 00       jle    1c <main+0x1c>
12:   f7 e9                   imul   ecx
14:   83 e9 01                sub    ecx,0x1
17:   e9 ed ff ff ff          jmp    9 <main+0x9>
1c:   c3                      ret
```

Výpočet faktoriálu: Sparc V8

```
00000000 <main>:
   0:   9d e3 bf 88     save  %sp, -120, %sp
   4:   a0 10 20 01     mov  1, %l0
   8:   80 a6 00 00     cmp  %i0, %g0
   c:   04 80 00 06     ble  24 <main+0x24>
  10:   01 00 00 00     nop
  14:   a0 5c 00 18     smul  %l0, %i0, %l0
  18:   b0 26 20 01     dec  %i0
  1c:   10 bf ff fb     b  8 <main+0x8>
  20:   01 00 00 00     nop
  24:   b0 10 00 10     mov  %l0, %i0
  28:   81 c7 e0 08     ret
  2c:   81 e8 20 00     restore
```

CPU (2/2)

- instrukce jsou zpracovávány v několika krocích:

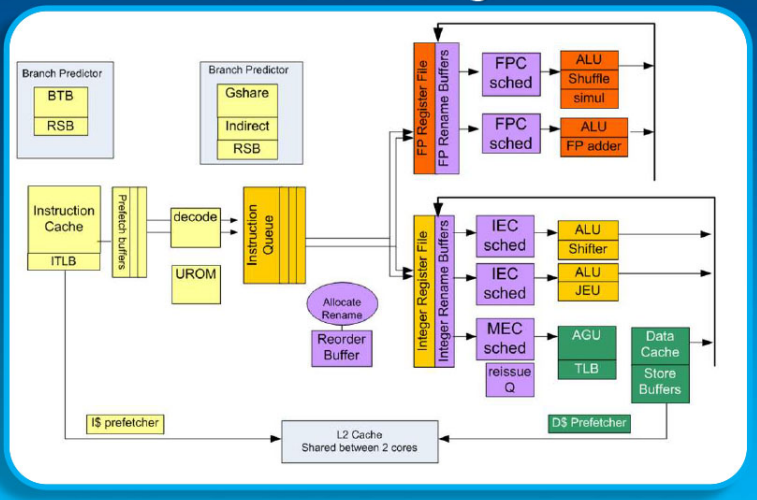
1. načtení instrukce do CPU (Fetch)
2. dekódování instrukce (Decode)
3. výpočet adres operandů
4. přesun operandů do CPU
5. provedení operace (Execute)
6. uložení výsledku (Write-back)

- pipelining – umožňuje zvýšit efektivitu CPU
- superskalární procesory – procesor může mít víc jednotek např. pro výpočty (FPU, ALU)
- je potřeba zajistit správné pořadí operací
- synchronizace, problém s podmíněnými skoky (branch prediction)
- Simultaneous multithreading (SMT): zpracování instrukcí více vláken v jednom cyklu

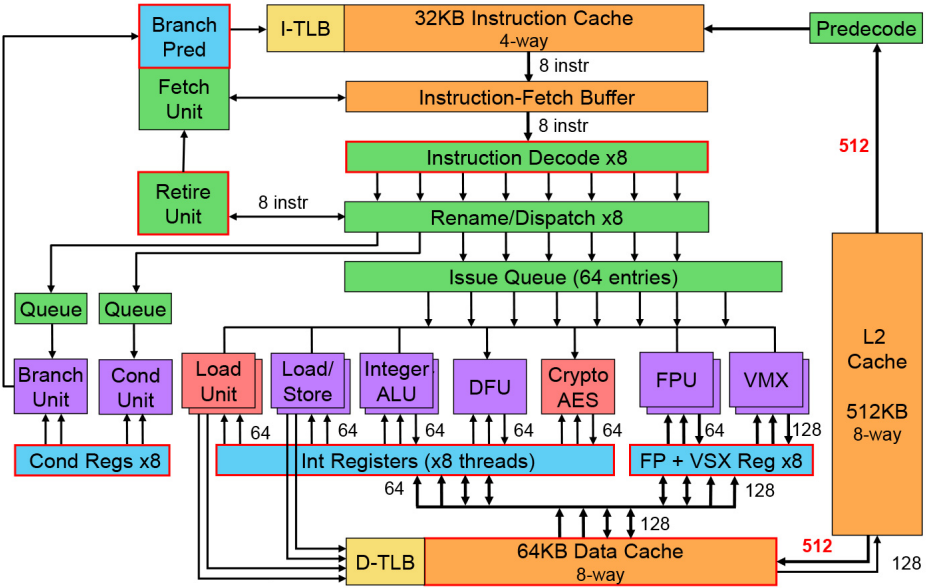
Pipeline

Instr. No.	Pipeline Stage						
1	IF	ID	EX	MEM	WB		
2		IF	ID	EX	MEM	WB	
3			IF	ID	EX	MEM	WB
4				IF	ID	EX	MEM
5					IF	ID	EX
Clock Cycle	1	2	3	4	5	6	7

Intel Silvermont



IBM Power8


Branch Pred
I-TLB
32KB Instruction Cache
4-way
Predecode
8 instr
Fetch Unit
Instruction-Fetch Buffer
8 instr
512
Instruction Decode x8
Retire Unit
8 instr
Rename/Dispatch x8
Issue Queue (64 entries)
Queue
Queue
L2 Cache
512KB
8-way
Branch Unit
Cond Unit
Load Unit
Load/ Store
Integer ALU
DFU
Crypto AES
FPU
VMX
64
64
64
64
64
64
128
Cond Regs x8
Int Registers (x8 threads)
FP + VSX Reg x8
64
128
128
D-TLB
64KB Data Cache
8-way
512
128

Architektura: Intel x86, resp. AMD64

- postupný vývoj od 16bitových procesorů (Intel 8086/8088), přes 32bitové (Intel 80386, 80486, Pentium) až po 64bitové (AMD64, x86_64)
- velká míra zpětné kompatibility
- emulace starších režimů
- kompatibilita na úrovni instrukční sady
- podobné instrukce (s novými modely přidávány další instrukce)
- rozdílné velikosti registrů

Intel 8086 (16bitové procesory)

- obecně použitelné registry jsou 16bitové
  - AX (Accumulator) – střadač pro násobení a dělení, vstupně-výstupní operace
  - BX (Base) – nepřímá adresace paměti
  - CX (Counter) – počitadlo při cyklech, posuvech a rotacích
  - DX (Data)
  - tyto registry lze rozdělit na dvě 8bitové části reprezentované jako registry AH, AL, BH, BL, ...
- DI (Destination Index), SI (Source Index) – přenost dat adresa cíle a zdroje
- BP (Base Pointer) – adresace parametrů funkcí a lokálních proměnných
- SP (Stack Pointer) – ukazatel na vrchol zásobníku (adresa vrcholu zásobníku)
- IP (Instruction Pointer) – ukazatel na aktuální místo programu, adresa instrukce následující za právě prováděnou instrukcí, není možné jej přímo měnit (jen patřičnými instrukcemi)
- F(LAGS) – příznaky nastavené právě proběhlou instrukcí

Intel 80386 (32bitové procesory)

- došlo k rozšíření 16bitových registrů na 32bitové
- pro každý 16bitový registr existuje jeho 32bitové rozšíření označené předponou E, tj. EAX, EBX, ECX, ESI, EDI, EBP, ESP, EIP, EF(LAGS)
- tyto registry spolu koexistují a sdílí místo
- obsahuje-li registr EAX hodnotu `0x12345678`, pak AX obsahuje `0x5678`, AL obsahuje `0x78` a AH `0x56`
- význam registrů zůstává stejný jako v případě 16bitové architektury

AMD64 (64bitové procesory)

- došlo k rozšíření 32bitových registrů na 64bitové
- pro každý 32bitový registr existuje jeho 64bitové rozšíření označené předponou R, tj. RAX, RBX, RCX, RSI, RDI, RBP, RSP, RIP, RF(LAGS)
- tyto registry spolu koexistují a sdílí místo (analogicky 16bitovým a 32bitovým registrům)
- význam registrů zůstává stejný jako v případě 16bitové i 32bitové architektury
- přibyly další registry R8, R9, . . . , R15, které mají svou 32bitovou část označenou jako RxD, 16bitovou část RxW a nejnižší byte je označen RxB (např. R8D, R8W, R8B)
- podle potřeby se kombinují 32bitové a 64bitové instrukce

Intel x86/AMD64: Operace (1/2)

- operandy instrukcí mohou být
  - `r` – registry
  - `m` – adresa místa v paměti
  - `i` – přímé hodnoty (konstanty), omezení na 32bitů
- pamět lze v jedné instrukci adresovat pouze jednou

```
MOV    r/m, r/m/i        ; op1 := op2
MOVABS r, i              ; op1 := op2  může být až 64bitová hodnota
ADD    r/m, r/m/i        ; op1 := op1 + op2
SUB    r/m, r/m/i        ; op1 := op1 - op2
NEG    r/m               ; op1 := - op1
IMUL   r, r/m            ; op1 := op1 * op2
OR     r/m, r/m/i        ; op1 := op1 | op2
AND    r/m, r/m/i        ; op1 := op1 & op2
XOR    r/m, r/m/i        ; op1 := op1 ^ op2
NOT    r/m               ; op1 := ~op1
```

Příklady

```
; do registru RAX ulozi obsah RBX
mov rax, rbx

; prevrati spodnich 16 bitu v registru ECX
xor ecx, 0x0000ffff

; pricte k registru cx hodnotu registru si
add cx, si

; takto nejde -- nesedi velikosti registru
add ecx, si

; vyneguje obsah registru edx
neg edx
```

Intel x86/AMD64: Operace (2/2)

```
INC r/m       ; op1 := op1 + 1
DEC r/m       ; op1 := op1 - 1

SHL r/m, i    ; op1 := op1 << op2 (neznaménková operace)
SAL r/m, i    ; op1 := op1 << op2 (znaménková operace)
SHR r/m, i    ; op1 := op1 >> op2 (neznaménková operace)
SAR r/m, i    ; op1 := op1 >> op2 (znaménková operace)
ROL r/m, i    ; rotace bitů doleva
ROR r/m, i    ; rotace bitů doprava
```

- místo přímé hodnoty (konstanty) lze použít registr CL

Registr příznaků

- jednotlivé operace nastavují hodnoty bitů v registru RF, EF(lags)
- ne všechny instrukce mění všechny příznaky
- registr EF mj. obsahuje příznaky:
  - SF (sign flag) – nastaven, pokud je výsledek záporný
  - ZF (zero flag) – výsledek byl nula
  - CF (carry flag) – nastaven, pokud při operaci došlo k přenosu mezi řády
  - OF (overflow flag) – příznak přetečení mimo daný rozsah hodnot
- některé další příznaky:
  - TF (trap flag) – slouží ke krokování
  - DF (direction flag) – ovlivňuje chování instrukcí blokového přesunu
  - IOPL (I/O privilege level) – úrověň oprávnění (2 bity, pouze jádro)
  - IF (Interrupt enable flag) – možnost zablokovat některá přerušení (pouze jádro)

Adresace paměti

- lineární struktura s pevnou délkou a náhodným přístupem
- přímá adresa – ukazuje na pevně dané místo v paměti
- nepřímá adresa – před přečtením hodnoty se vypočítá z hodnot registrů podle vzorce:

$$adresa = posunuti + baze + index \times factor$$

- posunutí je konstanta
- báze a index jsou registry
- factor je číslo 1, 2, 4, nebo 8
- kteroukoliv část vzorce lze vypustit

Adresace paměti v assembleru x86

- v assembleru se čtení/zápis do paměti zapisuje ve tvaru:
  `velikost [ ... ]`
- kde *velikost* může být (podle velikosti): `BYTE`, `WORD`, `DWORD`, `QWORD`

```
mov dword [rbx], eax
add ax, word [rbx + rsi * 2 + 10]
```

- pokud lze odvodit velikost dat z použitých registrů, je možné velikost vypustit

```
mov [rax], ebx
add ax, [rbx + rsi * 2 + 10]
```

- **Pozor!!!** `mov` **word** `[rax + rsi * 2 + 100], 42`

Vztah adresace paměti procesoru a jazyka C (1/2)

Dereference

```
mov eax, dword ptr [rbx]          ;; eax := *rbx
```

Pole

```
short *a = malloc(sizeof(short) * 10);
_asm {
    mov rbx, a
    mov ax, [rbx + rsi * 2]       ;; ax := a[rsi]
}
```

Vztah adresace paměti procesoru a jazyka C (2/2)

Strukturované hodnoty

```
struct foo { int x; int y; int z[10]; };
struct foo *a = malloc(sizeof(struct foo));
_asm {
    mov rbx, a
    mov [rbx], ecx                ;; a->x := ecx
    mov [rbx + 4], ecx            ;; a->y := ecx
    mov [rbx + rsi * 4 + 8], ecx ;; a->z[rsi] := ecx
}
```

Zarovnání hodnot (1/2)

- adresa paměti `mem` je zarovnaná na `n` bytů, pokud je `mem` násobkem `n`
- z paměti procesor čte celé slovo (např. 32 bitů) $\Longrightarrow$ vhodné, aby čtená hodnota ležela na zarovnané paměti (rychlejší přístup, snažší implementace CPU)
- některé CPU neumožňují číst data z nezarovnané adresy (RISC), jiné penalizují zpomalením výpočtu
- hodnoty jsou zarovnávány na svou velikost, např.
  - `char` na 1B,
  - `short` na 2B,
  - `int` na 4B, atd.
- tzn. hodnoty typu short jsou v paměti vždy na adresách, které jsou násobky 2, hodnoty int na násobcích 4, atd.
- velikost struktur se obvykle zakrouhluje na 4 nebo 8B (směrem nahoru)

Zarovnání hodnot (2/2)

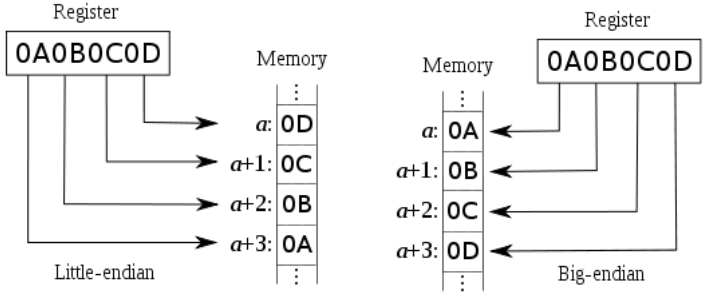
- Příklad:

```
struct foo {
    char a;
    /* mezera 3B */
    int b;
    char c;
    /* mezera 1B */
    short d;
};
```

- toto je chování překladače; lze jej změnit (je-li to nutné)

Uložení vícebytových hodnot: endianita

- liší se mezi procesory $\Longrightarrow$ potřeba brát v úvahu při návrhu datových formátů a protokolů
- **little-endian**: hodnoty jsou zapisovány od nejméně významného bytu (x86, Amd64, Alpha, ...)
- **big-endian**: hodnoty jsou zapisovány od nejvýznamějšího bytu (SPARC, IBM POWER, Motorola 68000, ...
- **bi-endian**: ARM, PowerPC, SparcV9, IA-64, ... (za určitých okolností lze přepínat)



Reprezentace hodnot

- čísla jsou v doplňkovém kódu (zápornou hodnotu dostaneme tak, že provedeme inverzi bitů a přičteme 1) $\Longrightarrow$ snadná manipulace
- znaménkové a neznaménkové typy (`unsigned int` vs. `int`)!!!
- pokud se hodnota nevejde do rozsahu typu $\Longrightarrow$ přetečení/podtečení

```
char a = 127 + 1;           // => -128
unsigned char c = 255 + 1;  // => 0
char b = -10 - 120;         // => 126
```

BCD (Binary Coded Decimal)

- čísla v desítkové soustavě 4b na cifru

Řetězce (1/2)

ASCII (American Standard Code for Information Interchange)

- způsob kódování znaků
- původně použité 7bitové hodnoty (později rozšířeny na 8 bitů)
- řídící znaky (CR, LF, BELL, TAB, backspace, atd.)
- národní abecedy – horní polovina tabulky, kódování ISO-8859-X, Windows-125X, atd.

Unicode

- znaková sada (definuje vazbu číslo $\Leftrightarrow$ znak)
- několik tzv. *rovin* po 65535 znacích (v současnosti 110.000+ znaků)
- první rovina se nazývá základní (Basic Multilingual Plane, BMP) – znaky západních jazyků

UCS (Universal Character Set)

- způsob kódování znaků Unicode
- pevně daná velikost
- UCS-2 – 16 bitů na znak, odpovídá základní rovině UNICODE
- UCS-4 – 32 bitů na znak, všechny znaky UNICODE

Řetězce (2/2)

UTF-8 (Unicode Transformation Format)

- kódování znaků s proměnlivou délkou
- zpětně kompatibilní s ASCII

bity	rozsah UNICODE	byte 1	byte 2	byte 3	byte 4	byte 5	byte 6
7	0000–007f	0xxxxxxx					
11	0080–07ff	110xxxxx	10xxxxxx				
16	0800–ffff	1110xxxx	10xxxxxx	10xxxxxx			
21	10000–1fffff	11110xxx	10xxxxxx	10xxxxxx	10xxxxxx		
26	200000–3ffffff	111110xx	10xxxxxx	10xxxxxx	10xxxxxx	10xxxxxx	
31	4000000–7fffffff	1111110x	10xxxxxx	10xxxxxx	10xxxxxx	10xxxxxx	10xxxxxx

UTF-16

- proměnlivá délka kódování
- rozšiřuje UCS-2
- varianty UTF-16BE or UTF-16LE
- Byte Order Mark (BOM) – umožňuje určit typ kódování (0xffef nebo 0xfeff)